
LDOS™

T A B L E O F C O N T E N T S

ACKNOWLEDGEMENTS

I - GETTING STARTED

USING THIS MANUAL

GENERAL INFORMATION

SYSTEM DEVICES

II - THE LDOS LIBRARY

LIBRARY COMMANDS: (* indicates extended library command)

APPEND	* ATTRIB	* AUTO	* BOOT	* BUILD
* CLOCK	COPY	* CREATE	* DATE	* DEBUG
DEVICE	DIR	DO	* DUMP	FILTER
* FREE	KILL	LIB	LINK	LIST
LOAD	MEMORY	* PURGE	RENAME	RESET
ROUTE	RUN	SET	SPOOL	* SYSTEM
* TIME	* TRACE	* VERIFY		

III - EXTENDED UTILITIES: BACKUP - CMDFILE - CONV - FORMAT - HITAPE
LCOMM - LOG - PATCH - REPAIR

DEVICE DRIVERS: JL - KI - RS232T

FILTERS: KSM - MINIDOS - PR

IV - JCL: THE LDOS JOB CONTROL LANGUAGE

LBASIC

V - TECHNICAL INFORMATION

VI - GLOSSARY

IN CASE OF DIFFICULTY

CUSTOMER SERVICE

WARRANTY

ADDENDUMS (IF ANY): QED

Second Edition LDOS 5.1.x Model III

A C K N O W L E D G E M E N T S

The following persons were members of the team that made LDOS possible:

Bill Schroeder
PROJECT LEADER

Roy Soltoff
SYSTEMS ANALYST

Chuck Jensen

Dick Konop

Tim Mann

Lance Micklus

The LDOS development team would like to thank the many people that provided suggestions and support during the creation of the LDOS products.

G E T T I N G S T A R T E D

Please observe the following steps exactly:

1) PLEASE FILL OUR AND RETURN THE WARRANTY CARD found in the rear of this manual. Especially important is the Name and Address information, which will assure you of receiving the quarterly newsletter, and any update information. If the WARRANTY card is not returned to LDOS Support Services within 30 days of purchase, your WARRANTY may be void.

2) Your LDOS master disk is WRITE PROTECTED with a small adhesive tab. DO NOT REMOVE THE WRITE PROTECT TAB.

3) Power up your TRS-80 system and all peripheral hardware. Place the LDOS Master diskette in drive :0 and press the RESET button to boot the LDOS diskette into the system. The LDOS logo will now appear on the screen. Enter in the correct date (mm/dd/yy), and the message LDOS READY will appear.

4) The first thing you should do is to make a BACKUP of your LDOS Master diskette. Follow the step by step procedures listed below.

All diskettes in the LDOS system will have a PACK ID assigned to them when they are FORMATted. This PACK ID consists of the disk Name and Master Password. During a Mirror Image BACKUP, the PACK ID's will be compared. If they are not the same, you will be asked whether or not to continue the BACKUP. Your LDOS Master Disk comes with the disk Name LDOS-5.1, and the disk Master Password of PASSWORD. The source disk PACK ID will be duplicated on the destination disk during a Mirror Image BACKUP.

FOR SINGLE DRIVE OWNERS:

Type in the command: **FORMAT :0 (STEP=3)**

The screen will clear and the LDOS disk FORMAT utility will be loaded. Two questions will appear on the screen. The correct response is shown for each question.

Question:	Your response:
DISKETTE NAME ?	**LDOS-5.1** <**ENTER**>
MASTER PASSWORD ?	**PASSWORD** <**ENTER**>

Answer the questions as shown. You will then see the message:

LOAD DESTINATION DISK AND HIT <ENTER>

At this point, insert a new, blank diskette in drive 0 and press <**ENTER**>. After the FORMAT is complete, this message will appear:

LOAD SYSTEM DISK AND HIT <ENTER>

Put the LDOS Master disk back in drive 0 and press <**ENTER**>. Now type in the command:

BACKUP :0 :0

The message INSERT SOURCE DISK (ENTER) will appear on the screen. Since your LDOS disk is the SOURCE disk, simply press <**ENTER**>. The message INSERT DESTINATION DISK (ENTER) will now appear on the screen. Put the disk you have just formatted into drive 0 and press <**ENTER**>. You will be prompted several times to swap the Source and Destination disks until the BACKUP is completed. At that point, the message INSERT SYSTEM DISK (ENTER) will appear. Place the BACKUP you have just made in drive 0 and press <**ENTER**>. The BACKUP is now complete.

FOR MULTIPLE DRIVE OWNERS:

Place a new, blank diskette in drive 1 and type in the command:

FORMAT :1 (STEP=3)

The screen will clear and the LDOS disk FORMAT utility will be loaded. Two questions will appear on the screen. The correct response is shown for each question.

Question:	Your response:
DISKETTE NAME ?	**LDOS-5.1** <**ENTER**>
MASTER PASSWORD ?	**PASSWORD** <**ENTER**>

Answer the questions as shown. LDOS will now FORMAT the disk in drive 1. When it is finished, the prompt LDOS READY will appear. To make the BACKUP, type in the command:

BACKUP :0 :1

LDOS will now make a BACKUP copy of itself on drive 1.

5) After the BACKUP of your LDOS disk is complete, remove the LDOS Master diskette from drive 0 and put it in a safe place. Be sure to leave it in its original jacket to protect it from dust and other contamination.

6) Label the BACKUP copy of the diskette as an original BACKUP of the LDOS Master diskette. You should use this diskette to make any other BACKUPs you require. Do not use the Master diskette except to create a BACKUP as just described.

7) It is extremely important that you now read the section entitled USING THIS MANUAL. That section contains information on the LDOS operating system as well as explanations of the User's manual layout and conventions.

8) If any questions or problems arise that cannot be solved by the User's manual, you will find a Toll Free user support number listed in the CUSTOMER SERVICE section and on the Warranty page, both in the rear of this manual.

LDOS FILE DESCRIPTIONS

This section will describe the various files found on your LDOS master diskette, and explain their functions. It will also describe how to construct a minimum system disk for running applications packages.

FILE GROUP - SYSTEM FILES (/SYS)

The descriptions of the /SYS files can be found in SYSTEM OVERLAYS near the end of the technical section.

FILE GROUP - UTILITIES

BACKUP - Used to duplicate data from one disk to another.

CMDFILE - A disk/tape, tape/disk utility for machine language programs.

CONV - Used to copy files from Model III TRSDOS to LDOS 5.1.x.

FORMAT - Used to put track, sector, and directory information on a disk.

LCOMM - A communications package for use with the RS-232 hardware.

LOG - Used to log in a double sided diskette in drive 0. Also updates the drive code table information the same as the DEVICE library command.

PATCH - Used to make changes to existing disk files.

REPAIR - Used to correct certain information on non-LDOS formatted diskettes.

FILE GROUP - DEVICE DRIVERS

JL - The LDOS JobLog driver program.

KI - The LDOS Keyboard driver, providing Type Ahead, Screen Print, and special <CLEAR> key functions.

RS232T - Used to configure and access the RS-232 hardware.

FILE GROUP - FILTER PROGRAMS

KSM - Establishes the KeyStroke Multiply feature, which allows assigning user determined phrases to alphabetic keys.

MINIDOS - Establishes certain immediate functions to shifted alphabetic keys.

PR - Allows the user to format printed output.

FILE GROUP - BASIC AND BASIC OVERLAYS

LBASIC - Enhanced Disk Basic program.

LBASIC/OV1 - Renumber overlay used with LBASIC's CMD"N" feature.

LBASIC/OV2 - Cross reference overlay used with LBASIC's CMD"X" feature.

LBASIC/OV3 - Error and Sort routines for LBASIC.

CREATING A MINIMUM CONFIGURATION DISK

All files except certain /SYS files may be removed from your run time drive 0 disk. Additionally, if the needed /SYS files are placed in high memory with the SYSTEM (SYSRES) command, it will be possible to run with a data diskette in drive 0 after BOOTing the system.

For operation, SYS files 1, 2, 3, 4, 8, and 10 should remain on drive 0 or be resident in memory. SYS 11 must be present only if any JCL files will be used. Both Libraries (SYS 6 and SYS 7) may be removed if no Library command will be used. SYS 5 and SYS 9 may be removed if the system DEBUGger is not needed. SYS 0 may be removed from any disk not used for BOOTing.

When using LBASIC, the OV3 overlay must also be present. OV1 and OV2 may be removed if no renumbering or cross referencing will be done.

The presence of any Utility, Driver, or Filter program is totally dependent upon the user's individual needs. Most of the LDOS features can be saved in a configuration file with the SYSTEM (SYSGEN) command, so the Driver and Filter programs won't be needed in run time applications.

The passwords for LDOS files are as follows:

System files	(/SYS)	Update password = **WOLVES**
Filter files	(/FLT)	Update password = **GSLTD**
Driver files	(/DVR)	Update password = **GSLTD**
Utility files	(/CMD)	Update password = **RRW3**
LBASIC		Update password = **BASIC**
LBASIC overlays	(/OV$)	Update password = **BASIC**
CONFIG/SYS		Update password = **CCC**

MASTER PASSWORD = **RS0LT0FF** (allows access to any file)

(Spelled with zeros, if your printer doesn't show the difference)

U S I N G T H I S M A N U A L

The LDOS User's manual is set up to be easily used and updated; It can be grouped into SIX distinct sections, each containing information on a specific group of operating instruction types. These sections can be identified by Section Identifier title blocks printed directly above the page numbers.

SECTION...1> is made up of general information about the LDOS system. It contains the TABLE OF CONTENTS, GENERAL INFORMATION, USING THIS MANUAL, and SYSTEM DEVICES. These four sections will give you an overall description of the abilities and standard operating conventions of the LDOS system.

SECTION...2> contains the LDOS LIBRARY COMMANDS. These commands are the heart of the operating system, and provide the link between the user and the computer. They are listed in alphabetical order, and each Library Command will have its own series of page numbers. New commands or changes to commands can easily be added to the manual in this manner. The LDOS manual will never become outdated.

SECTION...3> contains information on UTILITY, FILTER, and DEVICE DRIVER routines, provided for maintenance of your diskettes and programs, and to allow you to interface with other peripheral devices. They also add many of the enhancements that make LDOS the most versatile TRS-80 operating system.

SECTION...4> contains detailed operating instructions and information on the LDOS Job Control Language (JCL) and the enhanced LBASIC.

SECTION...5> contains detailed TECHNICAL information about the LDOS operating system, including important addresses and system routines. It will give programmers the information they need to build those "custom" routines that make applications programs function more rapidly and efficiently.

SECTION...6> contains the GLOSSARY, CUSTOMER SERVICE and WARRANTY information. When you purchased the LDOS operating system, you also obtained the best customer service arrangement in the micro industry. The LDOS Support Services group is committed to providing complete information and total assistance to all Registered LDOS owners. The Toll Free assistance number will be found in this section.

To locate the section of the manual you wish to access, refer to the Section Identifiers printed directly above the page numbers. All commands or programs in each section will be in alphabetical order. Each new command or logical division within a section will begin on the top of a right hand page.

Any time you encounter an unfamiliar word or definition, refer to either the GLOSSARY or GENERAL INFORMATION section of the manual to determine the exact meaning.

P A G E N U M B E R I N G

To provide a way of keeping your LDOS manual updated, the pages have NOT been numbered consecutively from the beginning of the manual. Instead, each section or command will start with page number 1. For example, the LIBRARY COMMAND section page numbering will show something like this:

APPEND - LIBRARY COMMAND
-1-

This would let you know that the page you are on deals with a LIBRARY COMMAND called APPEND. It is the first page for this command. Until you are familiar with the manual, you may wish to place marker sheets of some sort between the different sections. Tabs have been provided for major section divisions.

The user's manual will be updated each time there is a significant change in a particular operating procedure. If a change, correction or enhancement becomes available, it will be an easy matter to replace the appropriate pages without the need for complete reconstruction of the manual. It also makes it easier for the LDOS Support group to send out an immediate update or enhancement notice, without worrying about the cost of reprinting whole sections of the manual. This will assure you of receiving ALL relevant information as soon as possible.

C O M M A N D S Y N T A X A N D S T R U C T U R E

As you look through this manual, you should notice a very distinct structure regarding the command syntax of the LDOS system. Each Library Command, Utility, or Program section will begin with a very brief description of the function involved. Immediately following will be a "SYNTAX Block". This block will be laid out to show the command syntax, allowable parameters, and abbreviations, if any. A typical block might show the following:

```
============================================================
| THE COMMAND any files or devices (parameters)            |
|  "     "     "    "   "     "         "                 |
|                                                          |
| FILES/DEVICES DESCRIPTIONS                               |
|                                                          |
| PARAMETER DESCRIPTIONS                                   |
|                                                          |
| ABBREVIATIONS                                            |
============================================================
```

The first line(s) in the block will show the allowable COMMAND structure. Throughout this manual several words will be used as prepositions separating commands and/or parameters. They are:

TO	**ON**
OVER	**USING**

The use of these prepositions is ALWAYS optional; the LDOS command will function the same whether they are used or not. They are merely a convenience to allow the user to enter a command in more familiar syntax. If a preposition is NOT used, a space must be used in its place. NOTE: DO NOT use any of these prepositions as a filename - very unpredictable things will result!

The FILE/DEVICES description will describe the allowable full or partial File Specifications (filespec/partspec, and exclusion -filespec/-partspec), as well as denote if a Device Specification (devspec) may be used. It will also show the condition that the Specification must have for the particular command (active device, existing file, etc).

The PARAMETERS section will give a very short description of the allowable parameters dealing with the specific function. This description will generally be very brief, as a complete explanation will be given in detail in the text of that section.

Please note that many command parameters may have a default value if they are not specified. This may not be readily apparent, as many operating systems do not allow any parameters for these commands. One example is the PURGE command, with the parameter Q (Query). If the Q parameter is not specified, it will automatically default to Q=Y (Query each file before it is PURGEd).

The ABBREVIATIONS line will list all acceptable parameter abbreviations for the function. Note that (ON,YES and Y) and (OFF, NO and N) are completely interchangeable in most commands in the LDOS system.

U P P E R / L O W E R C A S E I N L D O S

The LDOS command interpreter has been endowed with an UPPER/LOWER case conversion feature. It is totally acceptable to enter any filespec, command or parameter in either upper OR lower case. If a specific command requires UPPER case only, it will be detailed in the description of that command. This manual shows all command lines and error messages in upper case. This is strictly an arbitrary convention. The actual LDOS messages will be displayed in UPPER and lower case.

G E N E R A L I N F O R M A T I O N

Your LDOS Disk Operating System is a user-oriented, device independent, totally supported package. LDOS provides compatibility between the TRS-80 Models I and III, so your data files and BASIC program files will be truly transportable. It also contains many features that have never before been found in a micro-computer operating system. It will be very easy for any owner of TRSDOS to step right up to the LDOS system, as most of the command structure and syntax is similar. However, to get the greatest value out of the system, it will be necessary to read and study the user's manual. This section will deal with generalized conventions that exist throughout the operating environment. It will also give an overall view of the total LDOS system.

Let's start by listing some of the hardware related features that you will find when using LDOS.

HARDWARE RELATED FEATURES

1) THE KEYBOARD will originally use the ROM keyboard driver. This will provide key debounce and key repeat. The debounce will prevent multiple characters from dirty or loose keyboard contacts. If a key is held down for about 1 second, the repeat feature will automatically repeat that key at the rate of 8 per second.

Note that the two <SHIFT> keys on the Model III can act separately. Therefore, certain functions will require the use of a specific <SHIFT> key. These functions will be indicated as needed in the manual by the use of <RIGHT or LEFT> along with the <SHIFT>.

The keyboard also has a built-in shift case reversal. Your keyboard characters will normally be entered in upper case. By pressing the <RIGHT SHIFT> and <0> keys together, any key pressed with the <SHIFT> held down will appear in lower case.

Control characters may be entered by using the <LEFT SHIFT> and <DOWN ARROW> keys as the control key. Simply hold down the shifted-down arrow and press the desired key.

The keyboard and KI/DVR

LDOS comes with a keyboard driver program called KI/DVR. The use of this driver is mandatory if functions such as Type Ahead, Screen Print, KSM, MiniDOS, LCOMM, and the SVC table are to be used. It is strongly recommended that the KI/DVR program with the TYPE option be active in your runtime system. It requires very little space and will make using LDOS even more pleasant.

Use of this keyboard driver will speed up the key repeat rate. It will also change the shift case reversal as follows. If typing in upper case, a shifted character will appear as lower case. If typing in lower case, a shifted character will appear as upper case. The <RIGHT SHIFT><0> keys will switch between the upper and lower case modes.

When using the KI/DVR program, the KSM and MiniDOS functions may also be used. Keys may be assigned special commands, functions, or characters with the KeyStroke Multiply (KSM) feature. These associated functions are then available when the <CLEAR> and desired unshifted key are pressed together. Due to this, it is necessary to press the <SHIFT> <CLEAR> to clear the screen. The MiniDOS filter program will give you immediate access to certain LDOS functions, such as a disk directory or free space display, a line printer top of form command, and a disk file kill command.

The <SHIFT> <BREAK> key will re-select a 5'' disk drive that has "timed out" and hung up the system. This may happen if a drive door was open, or if there was no diskette in the drive, etc. It should prevent having to reset the entire system. Do not use the <SHIFT> <BREAK> keys if the system is currently active!

2) THE DISK DRIVES can be 5'', 8'', or hard disk. LDOS will support double/single sides and density, and up to 80 tracks on floppy disks. Of course, you must have the appropriate hardware and drivers for this.

NOTE: We recognize that substantial savings can be made by purchasing non-Tandy disk systems for the Model III. We neither encourage or discourage this practice and will make every attempt to support your system regardless of the vendor that provides your drives. However, the floppy disk controller (FDC) must be completely Radio Shack compatible.

3) THE CASSETTE on the Model III can be used in either the 500 or 1500 baud mode. Use of the high speed (1500 baud) mode requires the use of the HITAPE/CMD driver program. Both the LBASIC and CMDFILE utilities allow high speed tape operation.

4) ALL SYSTEM HARDWARE DEVICES are totally independent of the normal routing structure found in most operating systems. Your system devices such as the video display and printer can be routed or linked almost anyway you could desire - to each other, to a disk file, to another device, etc. You can even create your own logical devices!

Once you have powered up your system, you can control the boot sequence to some extent. Note that if the <BREAK> key is held down during power up or reset, the computer will immediately enter ROM BASIC. After answering the date prompt, there are several keys that will modify the remaining boot sequence if held down. They are:

<CLEAR> This key will prevent any configuration file stored on the disk from being loaded. The configuration would have been created and stored with the SYSTEM (SYSGEN) Library command.

<RIGHT ARROW> This key will prevent the LDOS video driver from being loaded. The system will use the ROM video driver instead. This may be necessary for certain machine language programs. CAUTION: Using the ROM video driver will cause problems with Type Ahead, LCOMM, the Spooler, and any other LDOS function that uses interrupt processing, and should NOT normally be done!!

<D> This key will cause the DEBUGger (non-extended) to be loaded and executed. No configuration file will be loaded, and all memory above X'5200' will be untouched.

<ENTER> This key will prevent the execution of any breakable AUTO commands from taking place.

Once the system has displayed the message "LDOS READY", it is ready to accept a command from the user.

LDOS SYSTEM CONFIGURATION

Certain LDOS features can be configured by the user and stored in a disk file. They will automatically be loaded each time the system is powered up or rebooted. The Library command SYSTEM gives a description of the configuration procedure in its (SYSGEN) parameter section. Using the DEVICE library command will show the current system configuration, including active disk drives, device I/O paths, and some user selected options currently active. Once saved on disk, any configuration may easily be changed, deleted, or bypassed if the user desires.

FILES, DEVICES, AND LDOS

Throughout the manual, you will see references to "filespec" and "devspec". These are abbreviations for "file specification" and "device specification". (Refer to the Glossary for allowable filespecs and devspecs.)

Due to the device independence of LDOS, it is possible to interchange these two terms in most Library commands. For example, you can copy your keyboard to your line printer, or to a disk file. You can even append information from a device onto the end of a disk file! Each Library command will give detailed instructions and examples of the interchanging of filespecs and devspecs (if applicable).

Certain LDOS Library commands and Utilities allow the use of partspecs (partial filespecs). This will allow you to use the filename and extension fields to create groups of files with common filespecs, and then access these files as a group. LDOS creates "default" extensions in the filespec during some operations. Other operations will use these default extensions when searching for a file. Following is a list of LDOS default extensions along with suggestions for others that may help you "standardize" your file access.

ASM - The extension used by some Editor/Assembler programs for source files.

BAS - Possible extension for BASIC programs. Used by some BASIC compilers.

CIM - LDOS default for DUMP command. It stands for **C**ore **IM**age.

CMD - LDOS default for LOAD and RUN commands, and PATCH and CMDFILE utilities. Used to indicate load module format files.

COM - Used by some systems to indicate **COM**piled object code.

DAT - Possible extension for data files.

DCT - LDOS default for the SYSTEM (DRIVER) command (**D**rive **C**ode **T**able).

DVR - LDOS default for the SET command. Usually indicates a "**d**ri**v**e**r**" program.

FIX - LDOS default for files to be used by the PATCH utility.

FLT - LDOS default for files used with the FILTER command.

JBL - LDOS default for JOBLOG files.

JCL - LDOS default for the DO command. Stands for **J**ob **C**ontrol **L**anguage.

KSM - LDOS default for KSM Utility. Stands for **K**ey**S**troke **M**ultiply.

OVx - LBASIC extension for Overlay files (Overlay "x").

PCL - Used as a default by Electric Pencil for its text files.

REL - Used by some systems to indicate relocatable object code.

SCR - LDOS default for Scripsit text files.

SEQ - Possible extension for sequential files.

SPL - LDOS default for the SPOOL command.

SYS - LDOS SYStem files only. Do not use for your own files!

TXT - LDOS default for the LIST and DUMP (with the ASCII parameter) command. Stands for **T**e**XT** file.

LDOS COMMAND INTERPRETATION

The LDOS command interpreter will use the following conventions:

Drivespecs must always be preceded by a colon, whether used as part of a filespec or as a stand alone parameter.

The closing parentheses may be omitted from any of the LDOS commands.

All commands, filespecs, and devspecs entered in lower case will be converted to upper case before being acted upon. This results in all file names in directory entries being in upper case.

LDOS JOB CONTROL LANGUAGE

Through the use of the DO Library command and the LDOS JCL (Job Control Language), it is possible to create a sequence of commands and functions, store them in a disk file, and recall and execute them at any time. The JCL will allow for inputs, variables, and conditionals, and even has a screen flash and audio tone alert mode!

A JCL file is merely an ASCII file that contains a series of commands or JCL statements.

The JCL is an extremely versatile (and somewhat complex) feature of the LDOS system. It will be well worth your time to study the special JCL section and to experiment with the language until you become familiar with it. You will find your day to day operating becomes much easier with a few well thought out JCL files.

USE OF LOWER CASE

LDOS will permit the use of lower case for all command syntax and parameters, except where specifically noted in the manual. The screen messages and prompts will all appear in upper and lower case, even though they are shown as upper case only in the user's manual.

COMPATIBILITY WITH EXISTING OPERATING SYSTEMS

LDOS has tried to remain upward compatible with TRSDOS like operating systems. Most documented system calls have remained unchanged, and nearly all Basic programs should run unmodified. Total compatibility is not possible, as LDOS cannot control the design of operating systems released by other vendors. LDOS has made every effort to remain upward compatible with TRSDOS-like operating systems, and have provided many utilities and commands to transfer data and files between operating systems.

Assembly language programmers now have a Supervisory Call table available, which should guarantee future compatibility between LDOS-like operating systems.

To use files created on other operating systems, it is necessary to move them onto diskettes that have been FORMATted by LDOS. The LDOS utilities BACKUP, REPAIR, and CONV, along with the COPY, COPY (X), LOAD (X) and DUMP library commands will usually provide the means to transfer your program and data files onto LDOS formatted diskettes. Refer to the individual utility sections for compatible TRSDOS version numbers.

****** IMPORTANT ******

Under NO circumstances should you use files on other than LDOS FORMATted diskettes in your actual day to day operation. LDOS Support Services cannot support data manipulations on non-LDOS FORMATted diskettes!

Disks created under non-TRSDOS operating systems are not guaranteed to be compatible with LDOS.

NOTE: LDOS uses a pointer named HIGH$ to show the highest unused memory location. LDOS will never destroy information or utility routines you load into high memory as long as you set HIGH$ with the MEMORY Library command or protect memory when going into LBASIC. Programs that do not honor HIGH$ should not be used unless nothing of importance is to be kept in high memory.

The commands DEBUG (E), FILTER, LINK, ROUTE, SET, and SYSTEM, and drivers such as HITAPE do use high memory for certain functions. To see the current HIGH$ value, use the MEMORY Library command. If LDOS has not loaded any routines into high memory, HIGH$ will show X'7FFF', X'BFFF', or X'FFFF' for 16K, 32k, or 48K machines. LDOS always protects its own routines by decreasing the value in HIGH$.

USING BASIC WITH LDOS

Your LDOS diskette comes with an enhanced Disk Basic called LBASIC/CMD. This program contains most of the features of TRSDOS Disk Basic, plus many new features and enhancements. Two utility programs, LBASIC/OV1 and LBASIC/OV2, are included to provide renumbering and cross reference capabilities. Instructions for using all special LBASIC features will be found in the LBASIC section of the manual. For standard Disk Basic operating commands, please refer to a TRSDOS Disk Basic manual.

L D O S S Y S T E M D E V I C E S

The LDOS operating system is a truly "Device Independent" system. Each device the system uses has its own DCB (Device Control Block), and is totally independent of the hardware. Each device has its own driver routine, whether located in the BASIC ROM or in RAM memory.

An LDOS device is used or created by specifying an asterisk followed by two alphabetic characters. Your original LDOS Master diskette is configured with six devices in the device control table. To view these devices, use the DEVICE Library command. You will see the devices as listed:

*KI This is the Keyboard Input (your keyboard).

*DO This is the Display Output (your video screen).

*PR This is the PRinter output (your parallel printer).

*JL This is the Job Log (an output log of commands with a time stamp).

*SI This is the Standard Input (presently unused by LDOS).

*SO This is the Standard Output (presently unused by LDOS).

Note that these are just the LDOS system supplied devices; it is possible for the user to create HIS OWN DEVICES!

There is another LDOS device that is referenced in this manual, even though not shown in a "normal" device table. This device is the Comm Line (*CL), and will also be explained in this section.

The LDOS Device Independence makes it possible to ROUTE devices from one to another, or even to a disk file. It allows SETting the device to a totally different driver routine. It makes possible single or multiple LINKs of devices to other devices or to disk files. In other words, you may re-direct the I/O between the system, its devices, and the disk drives in almost any manner.

****** N O T E ******

Once a normal LDOS device has been pointed away from its original driver routine, it may be returned to its normal power up state with the RESET Library command. A user created device may be either disabled or completely removed via the RESET and KILL commands. Refer to these two commands for the exact and allowable parameters.

Besides just sounding impressive, this Device Independence feature has many PRACTICAL aspects. For example, the line printer is normally controlled by a very simple driver routine. However, the printer output may be FILTERed with the PR/FLT program supplied with LDOS.

This FILTER program allows you to set parameters such as the number of characters per line, the indent on line wrap around, the lines per page, and the page length.

If you don't have a printer, simply ROUTE the printer to a disk file and all printing will be saved, enabling you to print it later, on a system with a printer. You could also ROUTE the printer to the display, and see the characters appear on the video instead of going to the printer.

Throughout this manual, you will see reference to the terms "device" (or "devspec") and "logical device". These terms represent the six system devices PLUS any devices the user has created. To create a device, please refer to the Library commands LINK, ROUTE, and SET. It is possible to use certain Library commands involving data I/O such as APPEND, COPY, and KILL with device specifications (devspecs) as well as file specifications (filespecs). It is not possible to imagine or describe all situations involving the possible uses and creation of devices. What this section will do is to explain the six LDOS system devices. Any other device interaction or creation will be determined by the individual needs, sophistication, and imagination of the user.

*KI - The Keyboard

The *KI device is the keyboard. The normal *KI driver provides the keyboard debounce routine, as well as the key repeat and shift case reversal functions. You are advised NOT to ROUTE or LINK the *KI device unless you are EXTREMELY careful. You may inadvertently remove all input to the system or introduce totally unwanted characters. To send the *KI characters to a specific device or file, see the Library commands APPEND and COPY.

Several LDOS features do make use of the *KI device. They are the KI/DVR, KSM/FLT, and MiniDOS. These programs allow for such features as Type Ahead, Screen Print option, a KeyStroke Multiply option, and <CLEAR> key recognition for KSM and other functions. The address of the *KI driver routine will be changed to a location in high memory if any of these functions are used, or if *KI is ROUTEd, SET, or LINKed.

*DO - The Video Display.

The *DO device is the video display. The normal *DO driver address will be located in RAM, and can be displayed with the DEVICE Library command. If the DOWN ARROW key is held down during booting, it will prevent the LDOS video driver from being loaded and the system will use the ROM video driver instead. This may be necessary for certain machine language programs. CAUTION: Using the ROM video driver disables interrupts and will cause problems with Type Ahead, LCOMM, the Spooler, and any other LDOS function that uses interrupt processing, and should NOT normally be done!!

Note that the *DO driver routine address will also change if *DO is involved in a ROUTE, SET, or LINK.

If you wish to send a screen display to a disk file, there are some simple ways to accomplish this. You can ROUTE the printer (*PR) to a disk file, and then LINK *DO to *PR. This will send ALL screen displays (including errors - backspace characters, etc.) to the printer, which is ROUTEd to a file so the output will really go to the disk. You could also enable the screen print function and ROUTE the printer to a disk file. By pressing the <LEFT SHIFT><DOWN ARROW> and the <*> keys, the current screen display will be sent to the printer, and actually be written to a disk file. The *DO may also be LINKed to a disk file using a "phantom" user device (see the LINK Library command).

*PR - The Line Printer.

The *PR device is the line printer. This device may be SET to other drivers or ROUTEd to disk files very easily in the LDOS system. A printer FILTER program is supplied on your LDOS Master diskette, and is called PR/FLT. This program will allow you to set page size, line length, line indent on wrap around, and many other parameters. The operation of this driver routine is detailed in the FILTER section of the manual.

You may also use the SPOOL Library command to create disk and/or memory buffers to store information being sent to *PR and SPOOL it out as *PR becomes available (i.e. not in a BUSY state, such as printing a line).

***** NOTE *****

If *PR is ROUTEd to a disk file or the display, it is not necessary to have a line printer physically hooked to the system. All I/O to the printer will be sent to the appropriate device or file, and no lock up will occur.

*JL - The Job Log.

The *JL device is the system's Job Log. This unique feature will keep a log of all commands entered or received, along with the time they occurred. The time is determined by the setting of the Real Time Clock, and may be set or changed with the TIME Library command. To enable *JL, use the SET command to SET *JL to its driver program called JL/DVR (see the section on DEVICE DRIVERS). Every command or request processed through the LDOS command line interpreter will then be logged in the *JL file, along with the time of the request. You may also enter comments or data directly into the Job Log by using an LDOS comment line (any line beginning with a period), or by opening a Job Log file from BASIC and writing to it. The Job Log may be turned off by RESETting *JL, which will also close the associated disk file.

CAUTION: If the Job Log is routed to a disk file, it should NOT be SYSGENed.

*SI - The Standard Input.
*SO - The Standard Output.

The *SI and *SO devices are system generated devices provided by LDOS, although they are not presently used by the system. Both devices will initially be shown pointed (NIL). These devices are available to the user.

*CL - The Comm Line.

****** N O T E ******

This device has been designated *CL strictly for the purpose of standardizing examples throughout this manual, although any other available devspec could have been used (such as *RK, *CJ, *RS, *TM, etc).

The *CL device is the system's Communication Line. This device will allow characters to be sent and received using the RS-232 interface. The *CL will not normally be shown in the device table, but is always available to the user. To enable *CL, simply SET it to an appropriate driver program. There is an RS232T/DVR program supplied on your LDOS disk. Please note that the LCOMM/CMD program examples also uses *CL as its RS-232 link (see the LCOMM utility program).

Whenever I/O is needed via the RS-232 interface, the *CL will provide it. The RS-232 driver program allows *CL to interface between the LDOS system and external devices such as a serial line printer, an acoustic coupler (commonly called a MODEM), a hard wired telephone data set, a paper tape reader, etc. Please refer to the RS-232 Device Driver section for a complete description of the allowable configurations of the *CL lines.

T H E L D O S L I B R A R Y

Your LDOS operating system contains a set of commands which direct the overall operating environment. These commands are called LIBRARY COMMANDS. They interface between the extremely complex code of the operating system and the simple one line command entered by the user.

These LIBRARY COMMANDS are listed in the Table of Contents, and also in the Library Command LIB. They are divided into two groups, the PRIMARY and the SECONDARY (or Extended) Library Commands. Each of these Library sections resides in a different module of the operating system. In this manner, you may delete the module containing the particular Library Commands (if they are not needed), thereby freeing extra space on your diskette.

The Primary Library Commands are contained in the module SYS6/SYS. It may be deleted if none of the commands will be used during operation.

The Secondary (or Extended) Library commands are contained in the module SYS7/SYS. If the commands contained in the Extended Library are not needed, you may delete this module.

The next section of the manual gives detailed descriptions of all Library commands. The name of the command can be found directly above the page number on the bottom of each page. Those commands preceded by an asterisk are Extended Library commands.

A P P E N D
==========

This command lets you APPEND (add) one file onto the end of another. Its primary use is with data files or ASCII-type text files. Files that are in load module format such as "CMD" or "CIM" type files, cannot be APPENDed using APPEND. (To APPEND these types of files refer to the "CMDFILE utility" in the UTILITY section.) The syntax is:

```
============================================================
|  APPEND filespec1 TO filespec2 (STRIP)                   |
|  APPEND devspec TO filespec (ECHO,STRIP)                 |
|                                                          |
|  ECHO  is an optional parameter that will echo the       |
|        characters to the screen when APPENDing a         |
|        device to a file.                                 |
|                                                          |
|  STRIP is an optional parameter that will backspace      |
|        the destination file 1 byte before the APPEND     |
|        begins.                                           |
|                                                          |
|        filespec1 and filespec2 are valid LDOS file       |
|        specifications.                                   |
|                                                          |
|        devspec is any valid, active device capable       |
|        of generating characters.                         |
|                                                          |
|  abbr: ECHO=E                                            |
|                                                          |
============================================================
```

APPEND copies the contents of file1 onto the end of file2. File1 is unaffected, while file2 is extended to include the contents of file1. The files must each have the same Logical Record Length or the APPEND will be aborted with the error message "FILES HAVE DIFFERENT LRLS" and neither file will be touched.

For example, suppose you have two customer lists stored in data files WESTCST/DAT and EASTCST/DAT. You can add the WESTCST/DAT file onto the end of EASTCST/DAT file with the command:

```
APPEND WESTCST/DAT TO EASTCST/DAT
```

EASTCST/DAT will now be extended to include WESTCST/DAT, while WESTCST/DAT will remain unchanged.

You can also APPEND a device (capable of sending characters) to a file. For example:

APPEND *KI TO WESTCST/DAT

This command will cause characters that are input on the keyboard to be APPENDed to the file WESTCST/DAT. Depressing the <BREAK> key at any time will terminate this type of APPEND. Note that the keystrokes will not be shown on the display during this APPEND, as ECHO was not specified.

APPEND *KI TO WESTCST/DAT (ECHO)

This example will perform identically to the last one, except that any key typed will also be echoed to *DO (the video screen).

APPEND PAGE2/SCR TO PAGE1/SCR (STRIP)

This example would APPEND PAGE2/SCR to the end of PAGE1/SCR in the following manner. PAGE1 would be backspaced 1 byte, in effect allowing the first byte of PAGE2 to overwrite the last byte of PAGE1. This would be necessary when APPENDing files such as SCRIPSIT files that have an internal End of File marker in the file. If the STRIP parameter was not used, SCRIPSIT would load the APPENDed file only up to the first End of File marker, and ignore the APPENDed PAGE2 file.

*** A T T R I B**
==========

This command allows you to alter or remove the protection status of a file by changing passwords and/or the degree of access granted by a password. ATTRIB also allows the defining of whether a filename will be visible or invisible when a normal directory of the disk is displayed. ATTRIB will also allow you to alter the diskette name, Master password, and Lock or Unlock all Visible, non-SYStem files. The syntax is:

ATTRIB filespec.password:d (ACC=a,UPD=b,PROT=c,VIS/INV)
ATTRIB :d (LOCK,UNLOCK,MPW="aa",NAME="bb",PW="cc")

For filespec ATTRIBs, use the following parameters:

password = UPDate password, used only if a password already exists.

ACC= ACCess password

UPD= UPDate password

PROT= the PROTection level

VIS VISible file in directory

INV INVisible file in directory

abbr: ACC=A, UPD=U, PROT=P, VIS=V, INV=I

For disk ATTRIBs, use the following parameters:

:d is an optional drivespec, defaults to 0.

LOCK LOCKs all VISible non-SYStem files by changing the ACCess and UPDate passwords to the Master Password of the disk.

UNLOCK This parameter removes the ACCess and UPDate passwords from VISible, non-SYStem files.

MPW= Allows passing the disk's current Master Password in the command line.

NAME Allows changing the disk name.

PW= Allows changing the disk Master Password.

abbr: NONE

This section will deal with ATTRIBing a filespec.

The levels of PROTection associated with the passwords are as follows:

LEVEL	PRIVILEGE
EXEC	EXECute only
READ	READ, execute
WRIT	WRITe, read, execute
NAME	reNAME, write, read, execute
KILL	All access except re-ATTRIB
ALL	Allows total access
FULL	Same as ALL

The PROTection levels form a hierarchy, with the highest PROTection level (EXEC) allowing the least amount of ACCess.

When you create a file, the password you specify becomes both the access and update password. (If you don't specify a password, a string of 8 blanks is assigned as a default password for both access and update.)

The parameters UPD, ACC, PROT, VIS and INV may be abbreviated to their first character U, A, P, V, and I respectively. The levels of PROTection (abbreviated P) may be abbreviated to their first TWO characters; KI used instead of KILL, EX used instead of EXEC, etc.

The word which follows the "ACC=" is the ACCess password, and will grant ACCess up to and including the level of PROTection that is specified. The password that follows the "UPD=" is the UPDate password and always allows FULL access to the file.

If the VIS or INV parameters are not specified in an ATTRIB command, they will remain unchanged. If the file is currently VISible, it will remain so, and vice versa.

ATTRIB sets or changes the protection of a file which already exists on a disk. There are several ways to use this feature. Here are some examples of the use of ATTRIB:

```
ATTRIB CUSTFILE/DAT:0 (ACC= ,UPD=BOSSMAN,PROT=READ,VIS)
ATTRIB CUSTFILE/DAT:0 (A=,U=BOSSMAN,P=RE,V)
```

> This will protect the file CUSTFILE/DAT on drive 0 so that it can only be READ by a file read routine. No password will be required to open and READ the file because the ACCess password has been set to "null" by placing no password after "ACC=". It can't be changed or written to in any way unless the UPDate password (BOSSMAN) is used when specifying the file, in which case full ACCess would be given. Notice that the file will be VISible in the directory.

```
ATTRIB ISAM/BAS:0 (ACC=,UPD=SECRET,PROT=EXEC,INV)
ATTRIB ISAM/BAS:0 (A=,U=SECRET,P=EX,I)
```

After execution of this command no one will be able to LIST this program when it is brought into BASIC, because the PROTection level for the ACCess password has been set to EXECute only. The only way this file can be read into the computer is with "RUN" in BASIC (RUN "ISAM/BAS"). Notice no password is needed to RUN the program as none was set with the "ACC=" parameter of the ATTRIB command. This file cannot be LOADed, LISTed or PRINTed without using the UPDate password SECRET. FULL ACCess will be granted if the file is specified as ISAM/BAS.SECRET because the UPDate password has been given. The UPDate password will allow complete access regardless of the PROTection level that has been set. Notice that this file will be INVisible in the directory because the INV parameter has been specified.

EXAMPLE: RUN"ISAM/BAS"

Any attempt to look at this file after it is loaded and RUN will cause the program to be deleted from memory. LISTing or LLISTing the program cannot be done in the normal manner unless the program is loaded using the password SECRET.

```
ATTRIB ISAM/BAS.SECRET:0 (ACC=NOWAY,UPD=SECRET,PROT=EXEC,INV)
ATTRIB ISAM/BAS.SECRET:0 (A=NOWAY,U=SECRET,P=EX,I)
```

This command will do the same thing to this BASIC file except that now the only way to get the program into memory, even to RUN it, is to know the ACCess PASSWORD of NOWAY.

EXAMPLE: RUN"ISAM/BAS.NOWAY"

It will now be brought into memory and executed but it cannot be LISTed. Any attempt to interrupt the execution of the program will cause the program to be erased from memory.

```
ATTRIB ISAM/BAS SECRET:0 (ACC=,UPD=,VIS)
ATTRIB ISAM/BAS.SECRET:0 (A=,U=,V)
```

This command will get rid of all passwords and make the file ISAM/BAS visible in the directory. Notice that the UPDate password of SECRET was required to re-ATTRIB the file.

```
ATTRIB HOST/CMD:0 (INV)
ATTRIB HOST/CMD:0 (I)
```

This command will make the file invisible to the normal directory command DIR, without assigning any passwords to the file. To see an invisible file type DIR :d (I).

The following section deals with ATTRIBing a disk.

The ATTRIB command will allow you to change the disk name, the disk Master Password, and the PROTection attributes of all VISible and non-SYStem files. Any time the ATTRIB command is used, the disk's current Master Password must be supplied. You will be prompted to enter the current Master Password, if it is other than "PASSWORD" and not passed with the MPW parameter.

ATTRIB :0 (UNLOCK, NAME="MYDISK")

This command will remove all ACCess and UPDate passwords from the user's VISible non-SYStem files on drive 0. It will also change the disk's name to MYDISK. Since the current Master Password was not specified with the MPW= parameter, you will be prompted for it before this command is actually executed, if it is other than PASSWORD.

ATTRIB :1 (NAME="DATA",PW="SECRET",MPW="BOSSMAN")

This command will change the name of the disk in drive 1 to DATA. It will also change the Master Password to SECRET. Note that the current Master Password was specified as BOSSMAN with the MPW= parameter.

ATTRIB (LOCK)

This command will first prompt you for the disk's Master Password, if other than PASSWORD. It will then change the ACCess and UPDate passwords of all the user's VISible, non-SYStem files to the disk's current Master Password. This command will be carried Out on drive 0, as no drivespec was used.

ATTRIB :1 (NAME)

This command will first prompt you for the drive 1 disk's Master Password, unless it is PASSWORD. It will then prompt you for the new name to be given to the disk.

*** A U T O**
=========

The AUTO command lets you modify the power up sequence, by specifying a command to be executed immediately after power-up, reset or BOOT. The syntax is:

```
============================================================
|  AUTO *dos-command                                       |
|                                                          |
|    * is optional and if used will disable the ability    |
|      of <ENTER> to break the execution of the AUTO       |
|      dos-command and also disable the <BREAK> key.       |
|                                                          |
|    dos-command can be any executable LDOS command        |
|      with or without parameters up to 32 characters      |
|      in length.                                          |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

If the AUTO *dos-command has disabled the <BREAK> key, it is possible to re-enable the <BREAK> after the AUTO command has finished execution. See the SYSTEM (BREAK=ON) LIBrary command for complete instructions.

Here are some examples of the use of the AUTO command.

AUTO LBASIC

Will write the command LBASIC as an "AUTOmatic key-in" on the drive 0 diskette, replacing any previous AUTOmatic key-ins. From that point on, every time you power up using that LDOS diskette or press the reset button, LBASIC will AUTOmatically be loaded into memory and executed. An AUTOmatic key-in takes the place of a keyboard input, just as though the command had been typed in and <ENTER> had been pressed.

AUTO *DO INIT/JCL:0

After this has been written to the drive 0 disk, power-up or pressing the reset button will cause the DO file INIT/JCL:0 to be executed, which will allow several commands to be executed AUTOmatically (see DO command and JCL). Note the asterisk immediately preceding the command. This is optional; when used it will disable the ability of the <ENTER> key to break or halt the AUTOed command. The <BREAK> key will also be disabled from this point.

To restore the power up sequence to the normal LDOS READY, type:

AUTO

This will eliminate any stored automatic key-in by removing it from its storage place on the disk.

***** N O T E *****

You can override any breakable AUTO function during power up or reset by holding down <ENTER> during initialization. This may be your only way of regaining control of the system, if the dos-command is not a working program. If the AUTO function disables the <BREAK> key and the AUTOed program is non-functional, it may seem impossible to regain control of that disk. Should this occur, simply BOOT another (non-AUTOed) disk in drive 0. When the LDOS READY appears, place the non-functional disk in drive 0, type AUTO, and press <ENTER>. The runaway AUTO function will then be removed from that disk.

*** B O O T**
=========

This command causes the disk in drive 0 to be BOOTed into the system. It has the same effect as the reset pushbutton switch or a power up condition. The syntax is:

BOOT <CLEAR> <RIGHT ARROW> <ENTER> <D>	
Holding down the indicated key during the BOOT will result in the following actions:	
<CLEAR>	No SYSGENed configuration will take place.
<RIGHT ARROW>	The video driver used will be the ROM driver, rather than the LDOS driver.
<ENTER>	No breakable AUTO commands will be done.
<D>	The system DEBUGger will automatically be entered. Note that no SYSGENed configuration will be loaded.
abbr: NONE	

NOTE: Only double density diskettes may be used to BOOT the Model III LDOS operating system.

Holding down the <RIGHT ARROW> key during BOOTing will prevent the LDOS front end to the video driver from being loaded. The system will use the ROM video driver instead. This may be necessary for certain machine language programs. CAUTION: Using the ROM video driver will cause problems with Type Ahead, Lcomm, the Spooler, and any other LDOS function that uses interrupt processing, and should NOT normally be done!!

By typing this command, the LDOS system disk in drive 0 is BOOTed back into the system. All devices will be returned to their normal power up configuration as if the system had been turned off and then turned on again. Any required FILTERing, LINKing, ROUTING, SETting or setting of SYSTEM parameters must be done again at this point, unless a SYSTEM config file has been generated on drive 0 by the use of "SYSTEM (SYSGEN)" (See SYSTEM command). If the DATE has not been set, it will be asked for at this time. If the system has been SYSGENed, the user configuration will be loaded and executed at this time, and any AUTOed commands will be done.

*** B U I L D**
==========

This command allows the user to BUILD a file of desired character strings and save this file under any valid filespec. BUILD is in the system mainly to BUILD ASCII files for use with the DO, KSM and PATCH features of LDOS, although you may BUILD files containing any characters X'00 to X'FF' with the HEX option. The syntax is:

```
============================================================
|  BUILD filespec (HEX,APPEND)                             |
|                                                          |
|  filespec  is any valid LDOS filespec                    |
|                                                          |
|  HEX       optional parameter allowing a "packed"        |
|            HEXadecimal format only.                      |
|                                                          |
|  APPEND    optional parameter that allows APPENDing      |
|            the BUILD data to the end of existing files.  |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

The BUILD command is used to create a file (or APPEND to an existing file), a series of commands, comments, or character strings entered from the keyboard. If the filespec does NOT contain a /ext (extension), the system will automatically assign a "default" extension of /JCL, for Job-Control-Language (see DO and JCL). If a file with the identical name exists, the BUILD command will abort with the error message "File already exists", unless the APPEND parameter has been specified. If the APPEND parameter is used, the existing file must have been created with the BUILD command, or be in ASCII format.

Should the user wish to create a KSM type file (see KSM utility), the file extension should be /KSM. This will tell BUILD to prompt you with each key identifier as you enter what you wish that key to represent. This type of BUILD is detailed in the section on the KSM utility.

After the file has been opened, all characters that are typed will be placed in the file just as they appear on the video. Lines are limited to a length of 255 characters. Each line that is entered should be terminated by pressing the <ENTER> key. The BUILD will end and the file will be written to the disk when the <BREAK> key is pressed as the first character of any new line.

NOTE: No line in a JCL file should exceed 63 characters in length.

The HEX parameter will allow you to enter characters other than those directly available from the keyboard. Any one byte character value may be entered in the HEX format "nn". The line length during a HEX BUILD will be 254 characters, allowing 127 HEX notation characters to be entered. The HEX parameter uses a "packed" format, with NO spaces or delimiters between bytes.

For example, you could create a character string containing graphics characters in the following manner:

818A90A10D

This HEX format line contains the HEX bytes 81, 8A, 90, and A1. Note that the byte values are entered "packed" together, with no spaces or other delimiters between them. One of the possible uses for this format may be to build graphics strings to be used with the KSM function. If a file is to be used with the KSM function, do NOT embed the bytes 0D or 3B in the string, as these represent the Carriage Return and Semi-colon characters, unless you actually intend for these characters to be present. They will be acted upon by the KSM file as end of line (0D) and embedded <ENTER> character (3B). Note that each logical line must be terminated with a 0D. Therefore several "logical lines" may appear on each "physical line". Each logical line is terminated with a 0D in the entered string, and each physical line terminated by pressing <ENTER>. The <ENTER> does not terminate the logical line.

EXAMPLE: F50DF10DFA0D<ENTER>

This would represent three logical lines in a KSM type file. Notice the three 0D's in the string.

IMPORTANT: The HEX parameter will not cause the file to be stored in load module format; it will remain a normal ASCII image type file, even though some of the characters may be well out of the pure ASCII range.

When BUILDing files other than KSM or HEX, the line input length should be limited to 63 characters (for clarity). The BUILD will be terminated when the <BREAK> key is entered as the first character in a line.

Following are some examples of the BUILD command.

BUILD THISFILE:2

This will check drive 2 for a file named THISFILE/JCL. If it exists, a "File already exists" error will occur. Otherwise, the file will be opened on drive 2. Note that the default extension /JCL was used, as no extension was specified. All JCL files require less than 64 characters per line to be entered.

BUILD MYKEYS/KSM

This command will search all available drives for a file named MYKEYS/KSM. If the file exists, a "File already exists" error will occur. Otherwise, this file will be created on the first available drive. Since the extension was specified as KSM, the prompts A>, B>, C>, D>, etc. will appear one at a time so each of the alphabetic characters may be assigned the character string(s) they are to represent (see KSM). This BUILD will terminate after the letter Z, or when a <BREAK> is used as the first character of a line.

BUILD SPECIAL/:0

This will BUILD a file using the name SPECIAL with no extension. Using the "/" with no following characters is the only way to BUILD a file without an extension (defeating the default /JCL extension). Note that the file SPECIAL cannot already exist, or an error will be generated.

BUILD MYJOBS/JCL (APPEND)

This command would search all available drives for a file named MYJOBS/JCL. If not found, it would be created on the first available drive. If the file already existed, any input from the BUILD would be APPENDed onto the end of the file. This is the way, for example, to extend an incomplete JCL file.

BUILD DISPLAY/BLD (HEX)

This command would BUILD a file allowing the use of the "packed" HEX format. The file must not already exist, or an error will be generated. Information may be entered into this file as HEX bytes, and will be stored as a normal BUILD format file. This format will allow 127 HEX byte representations per physical line. Logical lines may continue on more than one physical line as long as a 0D does not appear, which would terminate the logical line. The <ENTER> is used to terminate a physical line.

If a non-hex digit is entered, the error message "BAD HEX DIGIT ENCOUNTERED" will be displayed, and the build will abort.

BUILD MYPROGA/FIX:0

This would BUILD a file with the desired extension of /FIX for use with the PATCH utility (see PATCH).

*** C L O C K**
==========

This command turns on or off the screen display of the real time clock. The syntax is:

```
============================================================
|  CLOCK (switch)                                          |
|                                                          |
|  switch  The switch ON or OFF                            |
|                                                          |
|  abbr: ON=Y, OFF=N                                       |
|                                                          |
============================================================
```

When you enter in this command it will activate a background task and display the "real time" clock in the upper right corner of the screen, at print locations 54 to 61. This clock is under software control and is fairly accurate. The clock will only run in the 24 hour mode, but will not automatically update the date every 24 hours. It can be established at LDOS READY using the TIME and DATE commands) or the TIME and DATE values may be POKEd in from LBASIC. (See TIME and DATE)

The "real time" clock will be turned off while the LDOS system is doing some of its disk I/O functions, like BACKUP and FORMAT. You will be notified of this by the message:

NOTE: REAL TIME CLOCK NO LONGER ACCURATE

This message notifies the user that the real time clock has lost several seconds or more, because the system had to turn off the hardware clock during certain critical functions and is no longer exact.

The CLOCK display may also be turned on and off with the <CLEAR><SHIFT><C> keys if the MiniDOS filter is active.

***** N O T E *****

It should be noted that the CLOCK in the upper right hand corner of the screen will take precedence over whatever BASIC may attempt to print at the screen locations occupied by the display.

C O P Y
=======

Copies data from one file or device to another file or device. The syntax is:

```
|  COPY filespec1 TO filespec2 (LRL=nnn,CLONE)              |
|  COPY filespec1 TO partspec (LRL=nnn,CLONE)               |
|  COPY filespec1 TO :d (LRL=nnn,CLONE)                     |
|  COPY filespec:d (X)                                      |
|  COPY devspec TO filespec (LRL=nnn,ECHO)                  |
|  COPY devspec TO devspec (ECHO)                           |
|  COPY filespec TO devspec                                 |
|                                                           |
|  LRL    is an optional parameter, where nnn= the          |
|         logical record length at which filespec2          |
|         is to be set (1 to 256).                          |
|                                                           |
|  CLONE  indicates the desire for an exact duplicate       |
|         of the directory entry of filespec1. All          |
|         ATTRIButes will be COPYed with the file.          |
|                                                           |
|  ECHO   will cause any characters copied from a           |
|         devspec to be echoed to the screen.               |
|                                                           |
|  X      is an optional parameter that will allow          |
|         a single drive copy from a non-system             |
|         diskette.                                         |
|                                                           |
|  abbr: LRL=L, CLONE=C, ECHO=E                             |
|                                                           |
```

The COPY command in LDOS is greatly enhanced and expanded over similar commands in other systems. The user of LDOS should become familiar with this important command as it is used in the LDOS system so the full power of this feature can be utilized. Special attention should be given to the ECHO, LRL and CLONE parameters, which are totally NEW to the COPY concept.

LRL is a parameter that allows the establishment of a new Logical Record Length for a file during the COPY process. If not specified LRL will default to the LRL of the file being copied. This can be very useful when restructuring data files and for changing ASCII type files to be compatible from one application to another. It may also be needed when converting a SOURCE file from one language to another to allow the file to be read by another application language.

CLONE provides a feature that has never before been available to the TRS-80 user. When the CLONE parameter is used, COPY will not only duplicate the contents of the file but will also duplicate the DIRectory entry. The ACCess and UPDate passwords will be COPYed as well as the assigned PROTection level, the VISibility, the CREATE flag, and the MODified status of the file.

If CLONE is NOT used, and an existing destination file was being copied over, the attributes of the destination file (except for the date) will be unchanged. If the COPY command creates a new file, any password included will become both the ACCess and UPDate password of the destination file, and the file will be VISible, even if the file it was copied from was INVisible. See ATTRIB for more on file ATTRIButes.

ECHO can only be specified when the source for the copy is a device. If specified, all characters will be echoed to the screen as they are sent to the destination file or device.

The X parameter provides a means of COPYing between two non-system diskettes. During this copy, the user will be prompted to switch disks as necessary. See the example of an X parameter copy at the end of this section.

How COPYs are DATEd in the DIRectory is also very important. Files that are COPYed with the CLONE parameter will not have their DATE touched. The same last-written-to date that appeared in the original will be moved to the CLONE-COPYed file. If the CLONE parameter is not specified, the DATE that was set as the SYSTEM DATE at BOOT will be written to the DIRectory as the last-written-to date for the COPYed file.

In all of the following examples, the use of the word TO between filespecs or devspecs is optional. Sped and spec2 need only be separated by a single space.

EXAMPLES OF COPYing: filespec1 TO filespec2
--

Note that when COPYing files, if filespec2 already exists on the destination drive, it will be overwritten by the COPY.

When COPYing files, the filename, extension, and password of filespec2 will automatically default to those of filespec1 if they are not specified. See the following examples.

```
COPY TEST/DAT:0 TO TEST/DAT:1
COPY TEST/DAT:0 TO /DAT:1
COPY TEST/DAT:0 TO TEST:1
COPY TEST/DAT:0 :1
```

These four commands will execute identical copies. All parts of filespec2 will default to those of filespec1 if not specified. The use of the word TO is optional in any COPY command.

```
COPY TEST/DAT TO :1
```

This command will search the disk drives until it finds a file named TEST/DAT and then COPY it onto drive 1, using the filespec TEST/DAT.

```
COPY TEST/DAT.PASSWORD:0 TO :1
```

This command would COPY the password protected file TEST/DAT.PASSWORD from drive 0 to drive 1. The file on drive 1 will be named TEST/DAT.PASSWORD. Remember that ALL parts of filespec2 will default to those of filespec1 if they are not specified.

```
COPY TEST/DAT:0 TO TEST/DAT.CLOSED:1
COPY TEST/DAT:0 TO .CLOSED:1
```

These commands will COPY the file TEST/DAT from drive 0 to drive 1. The file on drive 1 will be named TEST/DAT, and have the update and access passwords set to CLOSED. Notice that the second command dynamically assigned the name and extension of filespec1 to filespec2 and then added the password CLOSED. If a password exists on the file being COPYed it cannot be changed during a COPY. To change a password see ATTRIB.

```
COPY MYPROG/BAS.MYNAME:0 :1 (CLONE)
```

This will COPY the file MYPROG/BAS.MYNAME from drive 0 to drive 1. The CLONE parameter has been specified so both ACCess and UPDate passwords plus the MOD flag, the VISibility status and the PROTection level will be transferred with the file from drive 0 to drive 1. The last-written-to date in the original file on drive 0 will also be transferred.

```
COPY TEST/DAT:0 TO MYFILE:1
```

This command would COPY the file TEST/DAT from drive 0 to drive 1, with the file on drive 1 named MYFILE/DAT. Notice that the extension of filespec2 was not specified and defaulted to /DAT.

```
COPY TEST/DAT:0 TO MYFILE/:1
```

This command will COPY the file TEST/DAT from drive 0 to drive 1, with the file on drive 1 named MYFILE. There will be no extension on MYFILE because the "/" with no other characters was specified in filespec2.

```
COPY DATA/NEW:0 TO /OLD:0
```

This command will COPY the file DATA/NEW from drive 0 to a file named DATA/OLD on drive 0. The filename was not specified for filespec2 and defaulted to that of filespec1 (DATA).

COPY DATA/V56:0 TO DATA/V28:1 (LRL=128)

This command will COPY the file DATA/V56 on drive 0 to a file called DATA/V28 on drive 1. These two files will contain the same data but the logical record lengths will not be the same. We will assume that the original file had a record length of 256. This would be a normal Model III TRS-80 "random" type data file. The file DATA/V28 will be created by the COPY and will have a record length of 128 bytes. This ability to reset the LRL of a file is very useful when converting data to be used by a "BASIC" that can deal with BLOCKED files (record lengths less than 256), such as the BASIC you run with LDOS. This function is also necessary when you wish to APPEND two files but cannot because they have different logical record lengths. By COPYing one of these files and setting the LRL parameter to the desired length, the record length can be adjusted and the APPEND will then function.

COPY MANUAL/TXT.JWY:0 TO :1 (L=128,C)

This command will copy the file MANUAL/TXT with the password JWY from drive 0 to drive 1. In the process of doing the COPY the Logical Record Length (LRL) will be changed from whatever it was to 128. Note that the LRL parameter was abbreviated to L in the above example. The CLONE parameter was also specified using the abbreviation C, so all of the file's ATTRIButes and the file's DATE will be carried over in the CLONE mode, as described above.

COPY CONTROL:0 /ASC:1 (LRL=1)

This will COPY the file CONTROL to CONTROL/ASC on drive 1. The Logical Record Length of the file will be changed from whatever it was in CONTROL to 1 byte in length. This is an excellent way to convert a data file to a file that could be handled by a word processor (providing the data file was ASCII to start with).

EXAMPLES OF COPYing: devspec to devspec

When COPYing from devspec to devspec, it is very important that all devices specified be assigned and active in the system. Any ROUTEing or SETting that has been done to the devices may affect the COPY. Some caution is necessary when COPYing between devices, as non-ending loops can be generated and lock up the system. In other words, PLEASE do not involve devspecs in your copies unless you thoroughly understand the procedures and constraints involved. Destruction of files and/or locking up the system could easily result from lack of user understanding when using this complex structure of the COPY command.

Following are a few examples of possible devspec to devspec copies. The results produced by these copies may possibly be duplicated through the use of other LDOS commands (such as ROUTE and/or LINK).

COPY *KI TO *PR

This command will COPY the keyboard to the printer. As keys are pressed, they will be sent to the line printer. Depending on the printer, the characters may be printed immediately or may require that a linefeed/carriage return be sent before printing. The keystrokes will not be visible on the video because the ECHO parameter was not specified.

COPY *CL TO *PR (E)

This command will COPY *CL (the RS-232 Comm Line) to *PR (the line PRinter). Each character that is received by the RS-232 will be processed by the RS-232 driver and then presented to the line printer. Since the ECHO parameter (E) was specified, each character will also be echoed to the screen. Prior to executing this command, *CL must have been SET to an appropriate DRIVER.

EXAMPLES OF COPYing: filespec/devspec TO devspec/filespec

As with COPYing devspec to devspec the following examples are intended more for showing the vast abilities of the system rather than being considered as the only method of accomplishing a result.

COPY *KI TO KEYIN/NOW:0

This would allow the sending of all keyboard entries to the disk where they would be stored in a file named KEYIN/NOW. There is a catch here; because the characters that are typed are going directly to the file, they will NOT appear on the screen. To view the characters, specify the ECHO parameter. To terminate this COPY you should depress the <BREAK> key. The file will then be closed and LDOS Ready will appear.

COPY ASCII/TXT:0 TO *PR

This command will COPY the contents of the file ASCII/TXT to the line printer. Although this command is functional, it would give the same output as would LIST with the (P) parameter. The COPY in this example will terminate automatically when the end of the file is reached.

EXAMPLES OF COPYing with the X parameter:

The command COPY filespec:d (X) is similar to a regular COPY, except that the X parameter will allow transferring a file from one diskette to another without requiring an LDOS system present on either diskette.

The colon and drive number are optional so that you can choose to COPY a file on some drive other than drive 0. This command requires swapping diskettes several times in order to utilize the LDOS operating system modules to perform the transfer.

You will be prompted for the correct diskette and when to insert it into the drive doing the COPYing. The prompts are as follows:

> INSERT SOURCE DISK (ENTER) = The disk that contains the file to be COPYed.
>
> INSERT SYSTEM DISK (ENTER) = This is any LDOS SYSTEM diskette. If the diskette which is currently in drive 0 contains the complete system, just press <ENTER>.
>
> INSERT DESTINATION DISK (ENTER) = This is the diskette to receive the file. You must have enough space left on that diskette to contain the entire file to be COPYed. Under certain conditions, this prompt may appear twice in a row.

You cannot COPY (X) logical devices, only disk files. The disk files can be any type file made with any LDOS compatible operating system. Note that the source and destination disks MUST have different pack IDs (disk name and/or master password).

After the COPY (X), the destination disk file will have its ATTRIButes set as follows:

> The file will be VISible, whether the source file initially was or not.
>
> If a password is entered on the command line, the destination file will automatically have this password set as its UPDate and ACCess passwords. If no password is specified for the source or destination file, then the destination will have no password protection set.

The correct ATTRIButes for the destination file may be re-applied with the ATTRIB command.

*** C R E A T E**
==========

This command allows the creation of a file of the type and size that is requested by the parameters. The syntax is:

```
|  CREATE filespec (LRL=aaa,REC=bbbb,SIZE=cccc)              |
|                                                            |
|  LRL=  This is the Logical Record Length to be used.       |
|        It must be an integer in the range 1 to 256         |
|        (default LRL=256).                                  |
|                                                            |
|  REC=  This is the number of RECords of length LRL         |
|        to be allocated to the file.                        |
|                                                            |
|  SIZE=  This is the amount of space in K (1024 byte)       |
|         blocks that the file is to be able to hold.        |
|         SIZE should not be specified if LRL or REC are.    |
|                                                            |
|  abbr: LRL=L, REC=R, SIZE=S                                |
|                                                            |
```

The CREATE command is used to pre-CREATE a file of a specified type and size. This allows the file to be as contiguous as possible on the disk and limits the number of disk accesses that must be performed when dealing with the file. This pre-CREATE of a file also assures that the expected amount of space on the disk will be available for use by this file. This file will be dynamically expanded if the CREATEd size is exceeded, but it will never decrease in size below its current size. A file cannot be CREATEd that would require more space than is available on a disk. Remember, if a drivespec is not used, the system will attempt to create the file on the first available drive.

NOTE: if a file has been CREATEd, doing a DIRectory command with the (A) switch set will show:

For a CREATEd file - S: nnK
For a normal file - S= nnK

The normal equal sign (=) will be replaced by a colon (:) to indicate that the file length is the result of a CREATE rather than the actual size of the data in the file (see DIR command for a complete display example).

CREATE NEWFILE/DAT:0 (LRL=128,REC=100)

This command will CREATE a file named NEWFILE/DAT on drive 0. It will have enough space allocated to accommodate 100 records of 128 bytes each.

CREATE ASCII/DAT:2 (LRL=1,REC=5120)

This command will CREATE a file named ASCII/DAT in which records will have a length of 1 byte, and there will be space taken on the disk for 5120 (5K) of these one byte records.

CREATE MAIL/DAT:3

This command will CREATE the file MAIL/DAT on drive 3. There will be no space assigned to the file at this time. The file name is merely placed in the directory. This is very useful as it allows the placement of a yet-to-be-used file on a designated drive. Since not specified, the LRL of this file will be 256.

CREATE GOOD/DAT (REC=50)

This will CREATE a file named GOOD/DAT on the first available drive. There will be space taken for 50 RECords of 256 bytes each, since the default LRL is 256.

CREATE SMALL/FIL:1 (SIZE=1)

This command will CREATE a file SMALL/FIL on drive 1 and take 1,024 bytes of space for the file (in actuality 1 gran will be taken as this is the smallest unit of allocation the system can deal with).

IF THE FILE ALREADY EXISTS.....

CREATE INVENT/DAT (SIZE=20)

It is acceptable to CREATE a file that already exists. This is the manner in which you would permanently assign additional space to a file. If the SIZE, or the LRL times REC would cause this file to become smaller than it presently is, the CREATE will abort and the error message "File exists larger" will appear. The file will not be harmed.

*** D A T E**
=========

The DATE command is used to set the month, day, and year for use with your BASIC programs and by LDOS as it creates and handles your disks and your files. The syntax is:

```
============================================================
|  DATE mm/dd/yy                                           |
|  DATE ?                                                  |
|  DATE                                                    |
|                                                          |
|            mm - 2 digit month, 01 to 12                  |
|            /  - mandatory slash                          |
|            dd - 2 digit day, 01 to 31                    |
|            /  -mandatory slash                           |
|            yy - 2 digit year, 00 to 99                   |
|                                                          |
|  DATE ?    Date prompt at system power up.               |
|                                                          |
|  DATE      with no parameters specified will return      |
|            the current DATE.                             |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

It is more important to set the DATE with LDOS than other systems, because LDOS uses the DATE when creating and accessing files, and when making BACKUPs and FORMATting disks. If the DATE is not set, LDOS will make no date entries in the directory, when it would be useful to see the actual DATE. When looking at the directory you are able to see the DATE when a file was created or last written to. If the DATE was not set, this "last-written-to DATE" will NOT be updated, and the file would not show the true "last-written-to DATE". When creating new files, doing BACKUPs or FORMATting, LDOS will use 00/00/00 if the DATE has not been set. Because the DATE is so important in the LDOS system you will be prompted for it on power-up.

In the lower center of the screen at power-up the system will prompt for the DATE to be entered. You should answer this prompt with a valid DATE string. (Should you desire not to set the DATE just press enter at this prompt and the DATE will default to 00/00/00.) The prompt and the DATE you entered will then be erased and replaced by a display showing the day of the week in the standard three character abbreviation, the name of the month (also abbreviated), the day of the month, and the year expanded as 19xx.

It should be noted that LDOS will store two other numbers calculated from the current DATE. These are Day of the Year, and Day of the Week. These values will be placed in RAM memory, and will remain constant unless the DATE is reset. The Technical section details the exact locations and patterns of these values.

It should be noted that the DATE will stay "set" as long as the computer has power applied to it, providing the DATE storage area is not overwritten by user applications. Therefore when "RESETing" or executing the BOOT command after the DATE has been set, the system will automatically recover the DATE that was last set and will not bother with prompting for the DATE. The DATE that had been set will be displayed and the system will continue.

Should you wish to examine the DATE that is set in the system simply type DATE and press enter. If the DATE has been properly set, the system will return the currently set DATE in day-of-week, month, day-of-month, year format. The current DATE will also be sent to the Job Log, if active.

Should you wish to set the DATE to one other than the system is currently using, simply enter:

DATE mm/dd/yy

The new DATE will be set by the system.

EXAMPLE:

DATE 01/04/80

> Sets the DATE for the first month (January), the 4th day and the 80th (1980) year. At power-up the DATE is set to 00/00/00 and the system will prompt you for a DATE to be entered. Should the <ENTER> key be depressed as the response to this prompt the DATE will remain as at power-up, 00/00/00.

Because the "Real Time Clock" is turned off during certain I/O functions, the TIME and DATE may need to be corrected each day. Do not depend on the system clock for exact timing functions.

*** D E B U G**
==========

The DEBUG command turns the LDOS system's DEBUGing utility on or off. The syntax is:

```
+-------------------------------------------------------------+
|  DEBUG (switch,EXT)                                         |
|                                                             |
|  switch  is the switch ON or OFF. If not specified,         |
|          ON is assumed.                                     |
|                                                             |
|  EXT     turns on the EXTended DEBUGger                     |
|                                                             |
|  abbr:   EXT=E, ON=Y, OFF=N                                 |
|                                                             |
+-------------------------------------------------------------+
```

Unlike the other LIBrary commands, the DEBUG command does not immediately produce a visible result. It loads the system DEBUGger into memory and then waits to be activated. The EXTended DEBUGger also loads a separate block into high memory, and protects this area by decrementing HIGH$.

Once the DEBUGger has been turned on, it will be entered when one of the following occurs:

1) The <BREAK> key is pressed.

2) After a program has been loaded, before the first instruction in the program is executed. This includes LDOS DEVICE DRIVER programs and UTILITY programs.

The DEBUGger may also be automatically activated by holding down the <D> key during the bootstrap operation.

The DEBUGger will be disabled during the execution of any programs with an EXECute only PROTection status.

Refer to the following examples to turn the DEBUGger on or off.

```
DEBUG (ON)  Turns on the standard DEBUGger.

DEBUG (E)   Turns on the extended DEBUGger

DEBUG       Turns on the standard DEBUGger

DEBUG (OFF) Turns off the DEBUGger, standard and EXTended.
```

Once the DEBUGger is turned ON, it will remain active until it is turned OFF or until the system is BOOTed. Detailed examples of interaction between the DEBUGger and program modules will be given later in this section.

Following is a sample display of the DEBUGger screen.

```
AF = 0D 2C --1-1P--
BC = 0D 61 =>  79 9E 77 23 05 20 F9 C9  71 E5 D6 08 38 0E E1 E5
DE = 01 04 =>  1A 4D 45 4D 20 53 49 5A  45 00 52 2F 53 20 4C 32
HL = 00 54 =>  01 01 5B 1B 0A 00 08 18  09 19 20 20 0B 78 B1 20
AF'= 00 54 -Z-H-P--
BC'= 51 B0 =>  29 29 29 29 B5 6F CD 8A  51 20 EF 1F CE 81 C9 D6
DE'= 06 01 =>  09 28 42 FE 19 28 39 FE  0A C0 D1 77 78 B7 28 CF
HL'= 51 00 =>  02 C7 C6 02 FF CB 02 F7  10 32 E7 20 32 01 C7 43
IX = 40 15 =>  01 9C 43 00 9A 00 4B 49  07 C2 FE 31 3E 20 44 4F
IY = 00 00 =>  F3 AF C3 74 06 C3 00 40  C3 00 40 E1 E9 C3 9F 06
SP = 41 CA =>  52 04 C3 4B DD 03 15 40  5D 45 18 43 3F 3F 4C 00
PC = 00 62 =>  B1 20 FB C9 31 00 06 3A  EC 37 3C FE 02 D2 00 00
     3E04  =>  20 34 30 20 31 35 20 3D  3E 20 20 30 31 20 39 03
     3E14  =>  20 34 33 20 30 30 20 39  01 20 30 30 20 34 02 20
     3E24  =>  34 39 20 20 30 37 20 03  32 20 06 05 20 33 31 20
     3E34  =>  33 05 20 32 30 20 34 34  20 34 06 20 09 19 20 3D
```

The DEBUG display contains information about the Z-80 microprocessor registers. The display is set up in the following manner:

The register pairs are shown along the left side of the display, from top to bottom. The current contents of each register pair are shown immediately to the right of the register labels.

The AF and AF' pairs are followed by the current status of the flag registers to the right of the register contents. The other register pairs will be followed by the contents of the 16 bytes of memory they are pointing to.

The PC register will show the memory address of the next instruction to be executed. The display to the right of that address shows the contents of that address to that address + X'0F'.

The bottom four lines of the screen show the contents of the memory locations indicated by the address at the left of each line. Refer to the list of DEBUG commands for information on use of the DEBUGger.

Note that in ALL examples, any parameter dealing with an address or a quantity must be entered as a HEXadecimal number. The DEBUGger will not accept the backspace key. If an incorrect command has been entered it may be cancelled with the X key. Also, DEBUG only looks at the last four numbers entered in response to any address question.

In the following examples, the first line following a command will give the syntax of the command. There are 3 ways the DEBUG commands can be entered. The first requires the <ENTER> key be pressed. The second requires the <SPACE> bar be pressed. The third type is immediate and will execute whenever the command key is pressed as the first character in the command. The following commands are allowed in both the regular and the EXTended DEBUGger.

COMMAND: A

The command syntax is: A

This command will set the display to the alphanumeric mode.

COMMAND: C

The command syntax is: C

This command will single step through the instructions pointed to by PC (the Program Counter). If any CALL instruction is encountered, the routine called will execute in full. The destination of any jump instruction encountered must be in RAM memory, or the C command will be ignored.

COMMAND: D

The command syntax is: Daaaa<ENTER>

This command starts the memory display from the address aaaa.

COMMAND: G

The command syntax is: Gaaaa<ENTER>
Gaaaa,bkp1,bkp2

This command goes to a specified address and begins execution. The parameters are:

aaaa specified address. If omitted, the PC contents are used.

bkp1,2...optional breakpoint addresses. They will cause execution to stop at the specified breakpoint. One or both may be specified. The breakpoints must be in RAM memory. All breakpoints are automatically removed whenever you return to DEBUG.

COMMAND: H

The command syntax is: H

This command sets the display to show hexadecimal format.

COMMAND: I

The command syntax is: I

This command causes the DEBUGger to execute the command at PC and single-step to the next instruction. This command is identical to the C command except that any CALLS encountered will not automatically be executed in full; they must be stepped through instruction by instruction. Any jump or call instruction encountered must have its destination in RAM, or the I command will be ignored.

COMMAND: M

The command syntax is: Maaaa<SPACE>

This command will allow modification of a specified memory address aaaa. If the display is set to include the specified address, you will see vertical bars around that byte of memory. The address and current byte will appear in the lower left of the screen. To modify the byte,

enter in the desired characters. Pressing the (SPACE> bar will modify the byte and move to the next address. Pressing the <ENTER> key will modify the byte and exit from the M command. Pressing the <X> key will exit from the M command without modifying the current byte. If aaaa is omitted, the current memory modification address (shown by the vertical bars) will be used.

COMMAND: R

The command syntax is: Rrp dddd<ENTER>

This command will modify the contents of a specified register pair.

rp......represents the register pair to be modified.

dddd....represents the new register contents.

The contents of the registers can be seen while in the register display mode.

COMMAND: S

The command syntax is: S

This command changes the display format from the register display mode to the Full Screen mode. In the full screen mode, the display will contain 256 bytes of memory, with the current display address being contained in the display (see the D command). The register pairs will not be shown.

COMMAND: U

The command syntax is: U

This command will dynamically update the display, showing any active background tasks. It may be cancelled by holding down any key.

COMMAND: X

The command syntax is: X

This command will return the display to the normal register display mode.

COMMAND: ;

The command syntax is: ;

This command advances the memory display 64 bytes in the register mode and 256 bytes in the full screen mode.

COMMAND: -

The command syntax is: -

This command decrements the memory display by 64 bytes in the register mode or by 256 bytes in the full screen mode.

THE FOLLOWING COMMANDS ARE FOUND ONLY IN THE EXTENDED DEBUGGER.

EXTENDED COMMAND: B

The command syntax is: Baaaa,bbbb,n<ENTER>

This command will move a block of memory from one location to another.

aaaa...is the starting address of the memory block to move

bbbb...is the destination address of the memory block.

n.....is the number of bytes to move.

EXTENDED COMMAND: E

The command syntax is: Eaaaa<SPACE>

This command allows you to enter data directly into memory, starting at address aaaa. The contents of memory address aaaa will be displayed and you may then type in 2 HEX characters to replace the current contents. This operation will automatically advance to the next memory location, and allow you to continue entering characters until the <ENTER> key is pressed. If aaaa is omitted, the current memory modification address will be used.

EXTENDED COMMAND: F

The command syntax is: Faaaa,bbbb,cc<ENTER>

This command will fill a block of memory with a specified byte. The parameters are:

aaaa...The first address to be filled.

bbbb...The last address to be filled.

cc.....The specified byte to fill the locations with.

EXTENDED COMMAND: J

The command syntax is: J

This command will Jump over the next byte, in effect incrementing PC by 1.

EXTENDED COMMAND: L

The command syntax is Laaaa,dd<ENTER>

This command will locate the first occurrence of the byte dd, starting the search at address aaaa. If aaaa is not specified, the current memory modification address will be used. If dd is not specified, the last byte given in a previous L command will be used.

EXTENDED COMMAND: N

The command syntax is: Naaaa<ENTER>

This command will position the vertical cursor bars to the next load block. This instruction is used to move logically through a block of memory that has been loaded directly from disk using DEBUG. To use this instruction you must position the location bars over the file type byte at the beginning of any block. Press N<ENTER> and DEBUG will advance to the next load block header. This instruction may also be used by setting PC to the first byte of any forward JR instruction. Upon pressing N<ENTER>, PC will be incremented by the branching offset requested by the JR instruction, and the DEBUGger will remain in the command mode.

EXTENDED COMMAND: O

The command syntax is: O<ENTER>

This command is the normal way to return to LDOS READY.

EXTENDED COMMAND: P

The command syntax is: Paaaa,bbbb<ENTER>

This command will Print a block of memory from address aaaa to bbbb inclusively. The output will contain both HEX and ASCII formats in the following manner:

aaaa bb bb ... bb ccccccccccccccc

aaaa....represents the current line address.

bb bb...represents a line of 16 locations in HEX notation.

cccc....represents the ASCII equivalents of the locations.

EXTENDED COMMAND: Q

The command syntax is: Qii<ENTER>

This command will display the byte at the input port ii.

EXTENDED COMMAND: Q

The command syntax is: Qoo,dd<ENTER>

This command will send the data byte dd to output port oo.

EXTENDED COMMAND: T

The command syntax is: Taaaa<SPACE>

This command will allow you to type ASCII characters directly into memory, starting at address aaaa. The current contents of the address will be shown, and the command will wait for the next keyboard character. After the character is entered, you will advance to the next memory location. To exit this command, use the <ENTER> key. The <SPACE> character cannot be entered with this command. Pressing the <SPACE> will advance one memory location without changing the contents of the current location. If aaaa is omitted, the current memory modification address will be used.

EXTENDED COMMAND: V

The command syntax is: Vaaaa,bbbb,nn<ENTER>

This command will compare the block of memory starting at aaaa to the block of memory at bbbb. The compare will be for nn bytes. If the display is in the register mode, the first byte of memory displayed will be set to the first location in the block starting at aaaa which does not match the block at bbbb. The current memory modification address used by the M, E, and T commands will be reset to the corresponding byte in the second block.

EXTENDED COMMAND: W

The command syntax is: Waaaa,dddd<ENTER>

This command will search memory for the WORD specified with dddd. The search will start at memory location aaaa. If aaaa is not specified, the current memory modification address will be used. If dddd is not specified, the last word given in a previous W command will be used. The memory display will automatically be set to show the address where dddd was located.

EXTENDED COMMAND: DISK READ/WRITE UTILITY

The command syntax is: a,b,c,d,eeee,f

a is the desired disk drive number.

b is the desired cylinder.

c is the first sector to read or write.

d is the operation, R for Read, W for Write, * for a Directory Write.

e is the starting address in memory where the information read from the disk will be placed, or where information written to the disk will be taken from.

f is the number of sectors to read or write.

If the cylinder is not specified, the DIRectory track will be used. If the number of sectors is not specified, a single sector will be read. If the starting sector is not specified, a full cylinder will be read.

If an error is encountered during a disk function, the error number will appear on the screen, surrounded by asterisks. The error indication will repeat each time another error occurs. To abort the disk function, hold down the <ENTER> key.

D E V I C E (and Drive LOG-ON)
==========

This command will display all logical devices which are in use and the devices and files to which they are currently pointing and/or attached to. It will also "log on" the diskettes currently in the available disk drives by updating the Drive Code Table to show the number of sides, the density, and the location of the DIRectory track. The syntax is:

```
+-------------------------------------------------------------+
|  DEVICE                                                     |
|                                                             |
|  No parameters are required or allowed.                     |
|                                                             |
|  abbr: NONE                                                 |
|                                                             |
+-------------------------------------------------------------+
```

A typical device display might look like this:

```
:0   5" Floppy #1, Cyls= 40, Dden, Sides=1, Step=12 ms, Dly= 1 s
:1WP 5" Floppy #2, Cyls= 35, Sden, Sides=1, Step= 6 ms, Dly= 1 s
:2   8" Floppy #1, Cyls= 77, Dden, Sides=2, Step= 6 ms
:3   5" Rigid  #0, Cyls=153
*KI <=  X'FC11'
*DO <=> X'4DC2'
*PR <=> X'41E5'
*SI  =  Nil
*SO  =  Nil
*UD <=> TEXTFILE/TXT:1
Options: Type, JKL, KI/DVR, MiniDOS
System modules resident: 1, 2, 3, 8, 10
```

The information on the type and configuration of each drive in the system is found at the top of the DEVICE display. There are several fields in each line which explain what the system sees for disk storage. Here is a typical line of information pertaining to one drive and the explanation of the fields it contains.

```
:1WP 5" FLOPPY #1, CYLS= 40, DDEN, SIDES=1, STEP=6  MS, DLY= 1 S
---- -  ------ --  --------  ----  -------  ----------  --------
aabb c  dddddd ee  ffffffff  gggg  hhhhhhh  iiiiiiiiii  jjjjjjjj
```

aa This is the LOGICAL drive number the line deals with.

bb This is the diskette write protect status, with WP = Write Protected. See the SYSTEM (DRIVE=,WP) command.

cc This is the size of the disk, 5" or 8" will appear.

dd This is the type of drive FLOPPY or FIXED (hard) shown.

ee This is the PHYSICAL BINARY location of the drive on its cable. 1, 2, 4 or 8 will appear here. These numbers are the physical locations that refer to logical drive $, 1, 2 and 3 respectively (see SYSTEM (DRIVER) command).

ff This is the number of CYLinders (tracks) on the disk that was in the drive when it was last accessed.

gg This shows the DENSITY of the last disk accessed in the drive and will show either DDEN or SDEN (Double/Single DENsity).

hh This shows the number of sides on the last disk accessed by the drive and will be a 1 or a 2.

ii This shows the step rate in "MS" that the drive is set at.

jj This shows the DELAY time that will be imposed when accessing a 5'' minifloppy drive. The delay refers to the time the system will wait after starting the drive motor before it attempts to access the disk in the drive. It does NOT refer to the time the drive will stay on after an access.

The following briefly describes what each of the *xx devspecs refer to. For additional information see the section on DEVICES and DRIVERS.

*KI This device is the Keyboard Input which is controlled from the keyboard driver. This may be the ROM driver routine or the KI/DVR driver that allows type ahead, screen print, and <CLEAR> key recognition.

*DO This device is the Display Output (video).

*PR This device is the line PRinter.

*JL This is the JOBLOG control device which is shown NIL on power up, but must be SET to its driver to be used.

*SI This is the Standard Input device which will normally be pointed (NIL).

*SO This is the Standard Output and will normally be pointed (NIL).

*UD This may be any User created Device. It may be an asterisk followed by any two alpha characters. Setting up "phantom" or "real" devices is a very important part of the device independence of LDOS (see FILTER, LINK, ROUTE, and SET).

These device relationships and specifications will change from time to time depending on the use of the FILTER, ROUTE, SET, LINK, SYSTEM and RESET commands. Note that the *UD device has been ROUTEd to a disk file. The filename of the file is shown by the device that is providing the input to that file (see ROUTE). After you have ROUTEd or SET a device and/or driver you may enter the DEVICE command to see how the different devices have been affected. After any changes are made to the system, the DEVICE command will show where each of the hardware devices is going to enter the first driver dealing with that device, as well as the interaction between devices and/or files.

The DEVICE command will update disk information in the device table in the following manner. Each diskette in a currently enabled disk drive will be examined for number of sides, density, and location of the DIRectory track. If a drive is enabled but contains no diskette, the device table entry for that drive will not change. The LOG/CMD utility program will perform the same function, but for a single drive only.

If the DEVICE command should "hang up" or return totally incorrect drive information, the Drive Code Table does not contain the proper size and location of your drives. The physical location and size (5'' or 8'') must be correct for the DEVICE command to log on the drives. Should this problem occur, reBOOT LDOS with the <CLEAR> key held down. If using other than 5'' floppies, use the SYSTEM (DRIVE=,DRIVER) command to set the proper drive configuration for your system. After checking the drive information with the DEVICE command, save this configuration with the SYSTEM (SYSGEN) command.

The "Options:" line will show you the system options currently active. These options are established with the SYSTEM, SET, SPOOL, and FILTER commands.

You will also see which SYStem overlays are currently resident in high memory. See the SYSTEM (SYSRES=) library command for details on how to reside these overlays.

See FILTER, ROUTE, SET, LINK, RESET, SYSTEM and COPY for more detailed information on the devices and their drivers.

D I R
=====

This is the command which allows the examination of a disk DIRectory. Several parameters are allowed to set the type of data that will be displayed, as well as the class of file to be processed and where the output is to be directed (video or printer). The syntax is:

DIR :d (A,I,S,P,N,D="aa")
DIR filespec:d (A,I,S,P,N,D="aa")
DIR partspec with wcc:d (A,I,S,P,N,D="aa")
DIR -partspec with wcc:d (A,I,S,P,N,D="aa")

wcc WildCard Character <$> used as required for masking characters.

:d Optional drive specification.

S W I T C H E S

A Display the DIRectory in full Allocation format.

I Display the INVisible files.

S Display the System files.

P Direct output to the Printer.

N Non-stop display mode (will not pause after each 15 lines). Assumed if "P" is selected.

D= "M1/D1/Y1-M2/D2/Y2" will display those files whose MOD dates fall between the two dates specified, inclusive.

"M1/D1/Y1" will display those files whose MOD dates are equal to the date specified.

"-M1/D1/Y1" will display those files whose MOD dates are less than or equal to the date specified.

"M1/D1/Y1-" will display those files whose MOD dates are greater than or equal the date specified.

abbr: NONE

When requesting a DIRectory in the LDOS system, many options and switches are available to control the DIRectory output. This is very important in the LDOS system as LDOS supports 8" and HARD drives as well as double density and double sided. DIRectories on these larger drives can become very lengthy. To handle these massive DIRectories, LDOS has provided convenient methods of finding specific files or groups of files, even among hundreds of DIRectory entries. After entering a DIRectory command, the system will process the DIRectory request based on the format, content and switches set in the command line.

Output from the DIRectory command will be sent to the video display unless the "P" parameter has been specified, in which case the output will be sent to the line printer as well. As output is displayed to the video, the system will pause after it has displayed each screen full (15 lines) of data. To continue from this logical pause simply press any key. This logical pause will not occur if the "P" (for Printer) or "N" (for Non-stop) parameters are used. The logical pause will also be disabled if the DIRectory command has come from a JCL file executed by DO (see JCL and DO).

The D switch will allow you to specify a MOD date or group of dates to be used as criteria for file display. Files without dates will not be displayed if the D switch is set.

The S and the I switches are add-to switches. If S or I are not specified, only VISible, non-SYStem files will be processed. If you use the I switch, all VISible files as well as all INVisible files will be processed. If the S switch is used, all SYStem files in the DIRectory will be processed, along with all other VISible files. If S and I are both specified, all files will be processed. The S, I, A, N, D, and P switches may be used together in any combination. The P switch will automatically set the N switch option.

It should be noted that the absence of a drive spec will cause a "global DIRectory" to occur. Every active drive's DIRectory will be displayed, starting with drive 0. A "global DIRectory" may be interrupted by pressing the <BREAK> key.

Following are examples of the DIR command and its parameters. After each example of a DIR command, a typical display is shown along with an explanation.

```
DIR :2

      Free space=     13.8 K  Drive 2  LDOS-5.1 -- 04/21/81
      PATCH/CMD P           LCOMM/CMD           LOG/CMD
      SYSTEM/JCL            HARD/DCT            BASIC/OVN
      SCRIPSIT/FIX          PENCIL/FIX          BASIC/FIX
      MAIL/DAT +            MAIL/BAS +          MANUAL/PCL +
      PRINTER/ASM +         RS232L/DVR          PR/FLT
      RS232/DVR             KSM/FLT             BASIC/OVX
```

This is a simple DIR command on drive 2. Several things should be noted about the DIRectory display. The top line of all DIRectory displays will show the FREE space remaining on the disk in K (1024 byte block), followed by the drive number, the disk name and the date that the disk was created. Also note the "+" sign after four of the filenames in the display. These may appear on any DIRectory display, indicating that the filename that precedes it has been written to (MODified) since it was last BACKed UP. BACKUP deals with this MOD "flag" in very important ways (see BACKUP).

```
DIR :2 (A)

    Free space    13.8 K  Drive 2  LDOS-C   -- 12/03/80
    Filespec Attributes Prot / LRL #Recs / Ext  File Space  Mod Date
    PATCH/CMD P         EXEC / 256     9 / 1 S=     2.5 K  30-Nov-80
    MAP/CMD             ALL  / 256     3 / 1 S=     1.2 K
    SYSTEM/JCL          ALL  / 256     1 / 1 S=     1.2 K  29+Nov+80
    HARD/DCT            ALL  / 256     2 / 1 S=     1.2 K  29-Nov-80
    MAIL/DAT P +        READ / 128   100 / 1 S=    13.2 K  16-Jan-81
    MAIL/BAS +          ALL  /   1 10811 / 1 S=    11.0 K  16+Jan+81
    RS232L/DVR P        ALL  / 256      4/ 1 S=     1.2 K  29-Nov-80
    ------------------- ----   ---  ----  -- ------------  ---------
    aaaaaaaaaaaaaaaaaa bbbb   ccc  dddd  ee ffffffffffff  gggggggggg
```

This DIRectory command used the "A" (Allocation) parameter. When the "A" parameter is used, the DIRectory will appear as it does above. The FREE space, drive number, disk name and creation date will appear in the first line. The next line will be the heading line, containing descriptions of the columns below it. These columns, indicated by the lower case letters a-g, are further described as follows:

aa This field is the file name with its extension if one is present. If the file is password protected a "P" will follow the filespec. If the file is set to INVisible then an "I" will also appear following the filename. If the file has been MODified since it was last BACKed UP, a "+" sign (or MOD flag) will also appear in this field. If it is a SYStem file, an "S" will appear. S, I, P and "+" may appear together in any combination to show the file's actual status.

bb This field shows the PROTection level that has been set for the file.

cc This is the length of each logical record in the file.

dd This is the number of logical records in the file.

ee This is the number of "extents" (non-contiguous blocks of space) in which the file is written.

ff This is the amount of space in K (1024 byte blocks) that the file takes up on the disk. A "S:" indicates a CREATEd file, while a "S=" indicates a normally allocated file.

gg This is the date that the file was last-written-to. This date will be correct if the date was set during the last write to the file. If the date was not set then the system will not update the date and will change the "-" signs in the date to "+" signs, to show that the date is not current. If the file is NEW and has not been written to then this date will be the creation date. If the date was not set during creation then this field will be blank, until the file is written to while the date is set.

DIR /DVR:2

```
Free space=      13.8 K  Drive 2   LDOS-C   -- 12/03/80
RS232L/DVR P           TAPE/DVR            RS232/DVR P
SERIALPR/DVR
```

This command will display all files on drive 2 that have the extension of DVR. It will be much easier to handle your DIRectories if you use a consistent "scheme" for your extensions. It is suggested that standard extension conventions be observed. These may be found in the General Information section of this manual.

DIR -/DVR:2

```
Free space=      13.8 K  Drive 2   LDOS-C   -- 12/03/80
REPAIR/CMD             LOG/CMD             SCRIPSIT/FIX
LBASIC/CMD P           SCRIPSIT/LC         TESTFILE/DAT
PAGE1/SCR              PAGE2/SCR
```

This command will display all visible files on drive 2, except for any files with the extension /DVR.

DIR /SYS:2 (S)

```
Free space=      13.8  K  Drive 2   LDOS-C   -- 12/03/80
BOOT/SYS SIP            SYS6/SYS SIP         DIR/SYS SIP
SYS7/SYS SIP            SYS0/SYS SIP         SYS8/SYS SIP
SYS1/SYS SIP            SYS9/SYS SIP         SYS2/SYS SIP
SYS10/SYS SIP           SYS3/SYS SIP         SYS4/SYS SIP
SYS5/SYS SIP            SYS11/SYS SIP
```

This command will display all "SYStem" files and only the "SYStem" files. Notice that to accomplish this, the S (System) switch was set to enable the display of system files. If the S switch had not been set, then no files would have been displayed. If the /SYS extension had not been specified then the system files would have been displayed along with all other files on the disk.

```
DIR /SYS:2 (S,A)

      Free space=      13.8 K  Drive 2   LDOS-C   -- 12/03/80
      Filespec Attributes Prot / LRL #Recs / Ext File Space  Mod Date
      BOOT/SYS SIP        EXEC / 256     3 / 1 S=   1.2 K
      SYS6/SYS SIP        NO   / 256    60 / 2 S=  15.0 K   30-Nov-80
      DIR/SYS SIP         READ / 256     9 / 1 S=   2.5 K
      SYS7/SYS SIP        NO   / 256     5 / 1 S=   1.2 K
      SYS0/SYS SIP        NO   / 256    18 / 1 S=   5.0 K
      SYS8/SYS SIP        NO   / 256     4 / 1 S=   1.2 K
      SYS1/SYS SIP        NO   / 256     4 / 1 S=   1.2 K
      SYS9/SYS SIP        NO   / 256     5 / 1 S=   1.2 K
      SYS2/SYS SIP        NO   / 256     5 / 1 S=   1.2 K
      SYS10/SYS SIP       NO   / 256     3 / 1 S=   1.2 K
      SYS3/SYS SIP        NO   / 256     4 / 1 S=   1.2 K
      SYS4/SYS SIP        NO   / 256     5 / 1 S=   1.2 K
      SYS5/SYS SIP        NO   / 256     5 / 1 S=   1.2 K
```

This example displayed the same files as the previous example but they are displayed in the allocation format because the A switch has been set.

```
DIR BAS:2

      Free space=      13.8 K  Drive 2   LDOS-C   -- 12/03/80
      BASIC/OVN             BASIC/FIX            BASIC/OVX
```

This command will display all VISible files on drive 2, provided that the first three characters in the filename are an exact match with BAS.

```
DIR BAS:2 (I)

      Free space=      13.8 K  Drive 2   LDOS-C   -- 12/03/80
      BASIC/OVN             BASIC/FIX            BASIC/OVX
      BASIC/CMD IP
```

This command will do the same thing as the previous example with the exception that INVisible files will also be processed as the search for files that begin with BAS is taking place. This is why BASIC/CMD IP appears in the display for this example but not in the previous example's display.

```
DIR S:0 (A)

      Free space=     175.0 K  Drive 0   LDOS-41  -- 01/15/81
      Filespec Attributes Prot / LRL #Recs / Ext File Space  Mod Date
      SPEC/DIR +          ALL  / 256     7 / 1 S=    2.5 K  16-Jan-81
      SCRIPSIT/FIX        ALL  / 256     3 / 1 S=    2.5 K  16-Jan-81
      SB/CMD +            ALL  / 256     2 / 1 S=    2.5 K  18-Jan-81
      SYSTEM/JCL          ALL  / 256     1 / 1 S=    2.5 K  15-Jan-81
      SS/L                ALL  / 256    43 / 1 S=   12.5 K  15-Jan-81
      SHIFTB/CMD          ALL  / 256     2 / 1 S=    2.5 K  15-Jan-81
      SYS7A/FIX           ALL  / 256     1 / 1 S=    2.5 K  15-Jan-81
```

This DIRectory command displays all VISible files on drive 0 that begin with the letter "S". They are displayed in the allocation mode because the A switch has been set.

```
DIR $$S:2 (A)

     Free space=      13.8 K  Drive 2   LDOS-C   -- 12/03/80
     Filespec Attributes Prot / LRL #Recs / Ext File Space  Mod Date
     BASIC/OVN           ALL  / 256     7 / 2 S=    2.5 K  29-Nov-80
     BASIC/FIX           ALL  / 256    10 / 1 S=    2.5 K  29-Nov-80
     BASIC/OVX           ALL  / 256    10 / 1 S=    2.5 K  29-Nov-80
     MASS/DAT            READ / 128    90 / 2 S=   10.2 K  22-Dec-80
     TASKS/OVX           ALL  / 256    10 / 1 S=    2.5 K  15-Jan-81
```

This example shows the use of the "$" as a "wcc" or Wild Card Character. By using "$$S" as the desired filespec, all VISible files that have a third character of "S" in their filename will be displayed.

```
DIR -B$S/$$X:2

     Free space=      13.8 K  Drive 2   LDOS-C   -- 12/03/80
     BASIC/OVN             MASS/DAT             TASKS/OVX
```

This is a very complex example. It will exclude all files that have a B and an S as their first and third character, and also have a 3 character extension ending with X. Using the DIR listing from the previous example, you can see that BASIC/OVX and BASIC/FIX were excluded from the display.

When using the "wcc" mask it must be noted that all partspecs or partspecs containing wild card characters are treated as LEADING information. Therefore "$$$F" and "$$$F$" and "$$$F$$$$" would all yield the same results, regardless of the total length of the filenames involved. All files with "F" as the fourth character in their filename would qualify. Therefore "BA$$" is the same as "BA", as the presence of the "$" trailing "real" characters will not limit the size of the name that might match. Likewise for something like "$$$", this does NOT indicate only three character file names. Actually, specifying nothing but "$" characters is like not specifying anything, as all filenames will qualify.

```
DIR P/FIX:2 (A)

     Free space=      13.8 K  Drive 2   LDOS-C   -- 12/03/80
     Filespec Attributes Prot / LRL #Recs / Ext File Space  Mod Date
     PENCIL/FIX          ALL  / 256     5 / 1 S=    1.2 K  29-Nov-80
```

This command will show all files with the extension /FIX, providing that the first letter of the filename is P. The allocation format is used because the A switch was specified.

D O
===

The DO command executes user created JCL (Job Control Language) files. The syntax is:

DO character filespec (@LABEL,parm,parm...);
character is an optional DO control character $, =, *
filespec is a valid filespec - default extension /JCL.
@LABEL is an optional LABEL indicating a start point in the JCL file.
parm optional parameter(s) to be passed to the filespec program during execution.
; The optional semi-colon. When used, allows a DO command line greater than 64 characters.
abbr: NONE

NOTE: Please refer to the Job Control Language section of the manual for the creation of a JCL file, and for the allowable passing of parameters.

The DO command will compile and execute a series of commands that have been created by the user and stored in an ASCII disk file. The default file extension of the filespec is /JCL. No line in a JCL file may exceed 63 characters in length. The DO command will also pass optional parameters to the program being DOne.

The DO function is normally a two step operation - the compile and the execute. During the compile, a line is read from the specified file and then written to a file named SYSTEM/JCL. If this file does not exist, it will be placed on the first available drive. Once the line is compiled, it is then executed directly from the SYSTEM/JCL file. There must be at least one available (enabled and not write protected) drive in the system to compile and execute a JCL file. However, an execute only option is available with a DO control character, and will be explained later.

Please note that the occurrence of any error will terminate the DO execution. The <BREAK> key, if not disabled, will allow you to manually abort the DO.

The three control characters ($, =, *) will change the compile and execution phases of the DO command. When using these characters, a space character is mandatory between the word DO and the character. If the space is omitted, the character will be ignored. Note that if no character is specified, both the compile and execution will be done.

The @LABEL parameter will allow you to create JCL files with multiple entry points. Each entry point can indicate a different location at which processing will begin. NOTE: If the @LABEL function is used, the compile phase must be done or the DO will abort with an error message. If the @LABEL parameter is specified, the JCL file will be scanned WITHOUT execution up to the specified LABEL. Once the LABEL is reached, execution will begin and continue until the next LABEL, or until the end of the JCL procedure/file has been reached. The primary reason for the @LABEL parameter is to allow many different functions to be built into one large file. This will greatly conserve disk space, as a series of small JCL files would take up a minimum of 1 granule apiece. For complete definitions of JCL LABELs, refer to the JCL section of the manual.

If the @LABEL and parameters cause the DO command line to exceed 64 characters, the semi-colon character (;) will allow you to continue passing parameters once the DO has started. The proper use of the semi-colon is as follows:

1) Terminate the DO command line by enclosing as many parameters as you can in the parentheses. Close the parentheses, then insert the semi-colon character and press <ENTER>.

2) A question mark will appear on the screen. At this point, you may enter the remaining parameters, making sure they are enclosed in parentheses.

Refer to the following examples and descriptions as a guide to the uses of the DO function.

CHARACTER: $

The $ character will DO the compile phase only, without actually executing the commands. The DO will compile your JCL file to the SYSTEM/JCL file. This will test if the syntax of a new JCL file will compile properly.

CHARACTER: =

The = character will skip the compile phase and directly execute your JCL file. Be aware that some of the JCL features will be ignored if the compile phase is skipped. Refer to the Job Control Language section of the manual for a complete list of these features and limitations.

CHARACTER: *

The * character will rerun the last DO command that was compiled, by using the existing SYSTEM/JCL file. If this file does not exist, nothing will be DOne and an error message will be generated.

DO DRIVE/JCL

This command will compile and execute a file named DRIVE/JCL. The system will search the drives for a file named DRIVE/JCL and compile it to a file named SYSTEM/JCL, executing each line after it has been compiled.

DO = DRIVE/JCL

This command will execute the file DRIVE/JCL without compiling it to the SYSTEM/JCL file.

DO $ DRIVE

This command will compile the file DRIVE/JCL to the SYSTEM/JCL file. The file will not be executed. Note that the filespec DRIVE will use the default extension of /JCL.

DO MY/JCL:0 (@THIRD)

This command will compile and execute the program MY/JCL. All instructions in the program will be ignored up to the LABEL (@THIRD). Execution will begin at this point and will continue until the next LABEL or End of File is reached.

DO *

This command will execute the SYSTEM/JCL file. If the file does not exist, an error will be generated.

DO TEST/NEW:2 (D=5,E=6)

This command will compile and execute the file TEST/NEW on drive 2. The file will be compiled to the SYSTEM/JCL file and each line will be executed from this file. The parameters D=5 and E=6 will be passed as needed during the execution.

The following examples show what will happen if the space is omitted in a DO command.

DO=TEST/JCL

The use of the = character normally tells the DO command to skip the compile phase and directly execute each line of the JCL file. If the space between the DO command and the is omitted, the compile phase WILL BE DONE! This means that the TEST/JCL file will compile to the SYSTEM/JCL (creating the SYSTEM/JCL file if none exists).

DO$TEST/JCL

The $ character normally tells the DO to compile the TEST/JCL file without executing it. If the space between DO and the $ character is omitted, the execution WILL BE DONE!

DO*

This command will ignore the asterisk (*) and generate the error message FILE SPEC REQUIRED!

*** D U M P**
=========

This command DUMPs a specified block of memory to a disk file. The dump may be in load module or ASCII format. The syntax is:

DUMP filespec (START=,END=,TRA=,ASCII,ETX=)
filespec is any valid filespec - default extension /CIM
START= is the starting address of the memory block
END= is the ending address of the memory block.
TRA= is the transfer address or execution point.
ASCII is an optional parameter for an ASCII DUMP.
ETX= optional End of Text marker.
abbr: START=S, END=E, TRA=T, ASCII=A

The DUMP command DUMPs an exact image of the specified memory locations to a disk file in load module or ASCII format. The default file extensions are /CIM for non-ASCII dumps, and /TXT for ASCII dumps.

The following restrictions are placed on the DUMP command addresses.

START= The memory block must START above address X'5500'.

END= The ENDing address must be greater than or equal to the START address.

TRA= The transfer address may be any valid address. If not specified, the transfer address (TRA) will be back to the system.

Addresses may be entered in either decimal or HEXadecimal format. HEX addresses must be in the form X'aaaa'.

The ASCII and ETX parameters are used to dump the output in a pure ASCII file. Address loading information is not present in the file and the file cannot be loaded by the system loader. The file is identical to the file structure of most word processor systems such as SCRIPSIT or PENCIL. Following the last dump character, an End of Text (ETX) character is written. This character is normally an X'03', but may be changed with the ETX parameter to a character of your choice. For example, an Electric Pencil file will normally have an X'00' as the ETX character.

Here are some examples of using the DUMP command.

```
DUMP ROUTINE/CMD (START=X'7000',END=X'8000',TRA=X'7000')
DUMP ROUTINE/CMD (S=X'7000',E=X'8000',T=X'7000')
DUMP ROUTINE/CMD (S=28672,E=32768,T=28672)
```

These three commands will create identical files. The first two use HEX notation for the addresses, while the third is in decimal format. The results of these commands will be to DUMP the area of memory STARTing at X'7000' and ENDing at X'8000'. This block of memory will be DUMPed to a disk file named ROUTINE/CMD. If the file already exists it will be overwritten. If it does not exist, it will be created on the first available drive. The transfer address of the program will be X'7000'.

```
DUMP TEST:1 (S=X'9000',E=X'BC0F')
```

This command will DUMP the specified block of memory to a disk file named TEST/CIM. Since the file extension was not specified, it defaulted to /CIM. The transfer address was not specified and will be written to the file as a return to the system.

```
DUMP WORD/TXT:0 (S=X'7000',E=X'A000',A)
```

This command will dump the specified memory range to a disk file named WORD/TXT. Since the A (ASCII) option was specified, no address information will be written to the file, and the EXT (End of Text) character will be the normal X'03'.

```
DUMP WORD/TXT:0 (S=X'7000',E=X'A000',ETX=X'FF',A)
```

This command is identical to the last one except that the End of Text marker will be written as an X'FF'.

F I L T E R
==========

The FILTER command establishes a FILTER for a specified I/O path. The syntax is:

```
+-------------------------------------------------------------+
|  FILTER devspec USING filespec (parm,parm,...)              |
|                                                             |
|  devspec  any valid LDOS device                             |
|                                                             |
|  filespec the filespec of a FILTER program                  |
|                                                             |
|  parm     optional parameters for the filespec program      |
|                                                             |
|  abbr:    NONE                                              |
|                                                             |
+-------------------------------------------------------------+
```

The FILTER command is used to set an I/O path through a FILTERing routine contained in the specified program. Any data needed by the program may be passed via the optional parameters. The function of the FILTER program is to filter data as it passes between the specified device and its driver program.

A FILTER program can provide many useful functions during I/O processing. Lines and/or characters could be counted, with certain actions taking place when preset limits are reached. Character conversions could be performed, such as simply changing each linefeed to a null, or a complete conversion from ASCII to EBCDIC.

FILTER programs may be provided by the user. The ability to write FILTER programs will require knowledge of Z-80 Assembler. A complete description of FILTER routines and how they are written can be found in the technical portion of this manual, where several actual FILTER programs are shown.

Several FILTER routines are provided on your LDOS diskette. This example shows the use of the FILTER command with the LDOS FILTER program PR/FLT.

```
FILTER *PR USING PR/FLT (CHARS=80,INDENT=6)
FILTER *PR PR (C=80,I=6)
```

These two FILTER commands will produce identical results. Note that the use of the word USING is optional. Also, the default extension for the filespec is /FLT. This example will FILTER I/O directed to the line printer through the FILTER routine PR/FLT. The PR/FLT program is described in the Filter section of the manual.

As a result of the above FILTER routine, printed output will be limited to 80 characters per line. Also, any single line which is greater than 80 characters in length will wrap around, and be indented 6 spaces on the next line.

Another FILTER routine provided on your LDOS diskette is called KSM/FLT. It provides the KeyStroke Multiply feature of LDOS.

```
FILTER *KI USING KSM/FLT USING MYKEYS/KSM
FILTER *KI KSM MYKEYS
```

The above examples would produce identical results, and are illustrations of how to set up a KSM FILTER. The KSM feature will now be enabled, and would use the file MYKEYS/KSM to provide the KSM phrases. For complete information on KSM, refer to the Filter section of the manual.

```
FILTER *CL REMCR
```

This command would FILTER the *CL device's I/O using the FILTER routine found in a program called REMCR/FLT. For example, if *CL had been SET to the RS-232 driver, this command would FILTER I/O to and from the RS-232 port.

*** F R E E**
=========

This command will show the used and available space and files on each disk in the system. The syntax is:

```
+-------------------------------------------------------------+
|  FREE (P)                                                   |
|  FREE :d (P)                                                |
|                                                             |
|  P   an optional parameter that directs output to           |
|      the printer.                                           |
|                                                             |
|  abbr:  NONE                                                |
|                                                             |
+-------------------------------------------------------------+
```

To execute the free command simply type FREE at the LDOS Ready prompt. LDOS will respond with a display similar to this:

```
Drive 0 - LDOS5.1B 07/01/81  Files= 97/128, Space=   87/  180 K
Drive 1 - LDOS-5.0 06/01/81  Files= 39/ 64, Space    19/   88 K
-------   -------- --------  ------------  -----------  ------
aaaaaaa   bbbbbbbb cccccccc  ddddddddd eee  fffffffffff  gggggg
```

Several fields are displayed in each line, representing the FREE information about one disk. The line under the display contains alpha characters to show the fields that will contain the FREE information. The information given in each of the fields is:

aa This field shows the drive number that the rest of the information in the line will pertain to.

bb This field shows the name of the disk.

cc This field shows the DATE of creation of the disk.

dd This shows the number of DIRECTORY entries that are AVAILABLE for use (number of files that may be added).

ee This shows the TOTAL number of directory entries that the disk will support.

ff This shows the amount of FREE space in "K" (1024 byte blocks) that remains available for use on the disk.

gg This shows the TOTAL amount of space the disk will support in "K".

The FREE command without drivespec is global in its nature in that it will search all active drives, and may not be confined to a single drive. The FREE space available on a diskette is also shown in a DIRectory command.

Using the FREE command with a drivespec will bring up a FREE space map as shown below.

```
FREE :0
 Drive => 0     Size => 5"     Sides => 1     Density => DOUBLE
------------------------------------------------------------
 0- 6   x..     ..x     xxx     x.x     xxx     xxx     x.x
 7-13   x..     ...     ...     ...     ...     ...     ...
14-20   ...     ...     ...     DDD     xxx     .xx     xxx
21-27   xxx     xx.     xxx     xxx     xxx     xxx     xxx
28-34   ***     ...     ...     ...     xxx     xxx     xxx
35-39   xxx     xxx     ...     ...     ...
Free =>    82 K /   55 G / Name => LDOS5.1    Date =>  06/01/81
```

In this example, the disk used was a 40 track double density diskette. The top line will display information about the diskette size and type. The bottom line will show the amount of free space in both grans and K (1024 bytes), along with the disk name and date of creation.

The inner display area contains the details of the space allocation on the disk. The numbers on the left represent the cylinders. The grans per cylinder will be shown across each line, with 7 cylinders per line. This disk has 3 grans per cylinder, as it is a 5" double density disk. The grans per cylinder will vary according to diskette size, density, and number of sides.

A gran will be represented as one of 4 characters, explained here.

. (period) - will represent an unused gran.

***** - will represent a locked out gran.

X - will represent a used gran.

D - will represent a gran used for the Directory.

This display may also be sent to *PR by using the (P) parameter.

K I L L
=======

This is used to delete the specified file or device from the system. The syntax is:

```
+-------------------------------------------------------------+
|  KILL filespec                                              |
|  KILL devspec                                               |
|                                                             |
|  abbr: NONE                                                 |
|                                                             |
+-------------------------------------------------------------+
```

KILL only deletes the directory entry. The file is still present until the file area has been written over by the system. However, it is beyond the scope of this manual to explain how to recover KILLed files.

To deal with KILLing several files it is often easier to use the PURGE command in LDOS, which in effect is a controlled "MASS-KILL". This may be the case if the files to be KILLed contain a common filename or extension, as the PURGE command can deal with these files as a group. (See the PURGE command.)

Here are some examples of KILLing files:

KILL ALPHA/DAT:0

This command will KILL the file named ALPHA/DAT that is present on drive 0. After execution of this command the file and the data in it will no longer be accessible to the system, so KILL carefully.

KILL DELTA/DAT

This command will KILL the file DELTA/DAT on the first drive that it is on. Be careful! Without a drivespec you could KILL the file where you did not intend to.

KILL MIDWEST/DAT.SECRET:0

This command will KILL the password protected file MIDWEST/DAT.SECRET on drive 0, as long as SECRET is the proper password. If the file's ATTRIButes include a PROT level of NAme or higher, SECRET must be the UPDate password to kill the file. If the proper password is NOT supplied, an error will be generated and the file will not be killed.

The LDOS system does NOT permit the KILLing of any devspec that is a normal power-up device. These devices are *JL, *KI, *DO, and *PR. Attempting to KILL these devices will produce an error message, and the KILL will abort.

However, any other devices may be KILLed, as long as they are RESET first. The command 'KILL devspec' will result in the devspec being completely removed from system device space.

KILL *CL

This command will in effect make *CL (the Comm Line) disappear from the system device control table, assuming the *CL was RESET before the KILL was done.

***** N O T E *****

KILLing of logical devices is an advanced feature of the LDOS system and as such should be treated with respect as undesirable results can occur. If the user does not completely understand "DEVICE INDEPENDENCE", the KILLing of devices should not be done!!

L I B
=====

This command will display the LDOS command LIBraries. The syntax is:

```
============================================================
|  LIB                                                     |
|                                                          |
|  no parameters are needed                                |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

After execution of this command, the LDOS command LIBraries will be displayed as shown below.

LIBRARY <A>

APPEND	COPY	DEVICE	DIR
DO	FILTER	KILL	LIB
LINK	LIST	LOAD	MEMORY
RENAME	RESET	ROUTE	RUN
SET	SPOOL		

LIBRARY <B>

ATTRIB	AUTO	BOOT	BUILD
CLOCK	CREATE	DATE	DEBUG
DUMP	FREE	PURGE	SYSTEM
TIME	TRACE	VERIFY	

LIBrary <A> is the primary LDOS command LIBrary, and is located in the SYS6/SYS system module. LIBrary <B> is the secondary command LIBrary, and is located in the SYS7/SYS system module. You may delete either system module containing the LIBraries if the commands included in it will not be used.

The secondary LIBrary commands are indicated throughout this manual by an asterisk preceding the command. This asterisk can be seen in the command name directly above the page numbers, and on the first page of each LIBrary command

L I N K
=======

This command LINKs together multiple logical I/O devices. The syntax is:

```
+-------------------------------------------------------------+
|  LINK devspec1 TO devspec2                                  |
|                                                             |
|  devspec is any currently enabled logical device            |
|                                                             |
|  abbr: NONE                                                 |
|                                                             |
+-------------------------------------------------------------+
```

This command is used to LINK together two logical devices. Both devices must be currently enabled. Once LINKed, any output sent to devspec1 will also be sent to devspec2. Any input requested from devspec1 may also be supplied by devspec2. Note that the use of TO between devspecs is optional, and only a space is actually required.

The user is cautioned about making multiple LINKs to the same device(s), as it is possible to create endless loops and lock up the system.

The order of the devices in the LINK command line is important, since output to devspec2 will not be sent to devspec1, nor can input requested from devspec2 be supplied by devspec1. Also, ROUTEing devspec1 will destroy its LINK to devspec2, but ROUTEing devspec2 is perfectly acceptable.

Once LINKed, devices can be un-LINKed by the command 'RESET devspec'. A global RESET or a reBOOT will also un-LINK devices. See the RESET and BOOT commands for further information.

Following are some examples of the use of LINK.

LINK *DO *PR

This command will link the video display to the line printer. All output sent to the display (devspec1) will also be sent to the line printer (devspec2). Once linked, the line printer MUST be enabled if it is physically hooked to the system (i.e. the cable is connected to both the printer connector and the printer). If the printer becomes de-selected or faults (out of paper, etc.) the system will hang up. Remember that both LINKed devices must be enabled. Note that any output sent individually to the printer, such as an LPRINT from Basic, will NOT be shown on the video display.

LINK *PR *DO

This command will LINK the line printer to the video display. All output sent to the printer will also be sent to the video display. The line printer must be on line and enabled if any printing is to be done. This LINK will not send any characters from the video to the line printer.

Although files may not be directly LINKed to a device, it is still possible to accomplish the same results through the use of "phantom" devices. This example will show how to accomplish a devspec TO filespec LINK.

Suppose you wish to LINK your line printer to a disk file named PRINT/TXT on your drive 0 diskette. Follow the steps below.

STEP 1) A "phantom" device must be created. For this example we will create a device named *DU. To do this, use the ROUTE command in the following manner:

```
ROUTE *DU TO PRINT/TXT:0
```

This will create a device named *DU and ROUTE it to a disk file named PRINT/TXT on drive 0. If the file does not exist, it will be created and dynamically expanded as needed. If the file already exists, any data sent to the file will be APPENDed onto the end of the file.

STEP 2) The printer can now be LINKed to the file with the following command:

```
LINK *PR *DU
```

The printer is now linked to the device *DU, which in turn is ROUTEd to the disk file PRINT/TXT. All output sent to the line printer will also be sent to the device *DU (in effect, written to the disk file PRINT/TXT).

Please note that the file PRINT/TXT will remain open until a RESET *DU is done. If you wish to break the LINK between the printer and the file without closing the file, then use the command RESET *PR. For further information, please refer to the ROUTE and RESET commands.

L I S T
=======

The LIST command will send a LISTing of a file to the video display or line printer. The syntax is:

LIST filespec (NUM,HEX,TAB,P,LINE=aa,REC=bb,LRL=cc)
NUM Sets line NUMbering mode for ASCII text.
HEX Sets HEXadecimal output format.
TAB Sets TAB expansion for ASCII text.
P Directs output to the line printer.
aa LINE in text file where ASCII LIST is to begin.
bb RECord number where HEX LIST is to begin.
cc The Logical Record Length to be used to display a file when in the HEX mode.
abbr: NUM=N, HEX=H, TAB=T, REC=R, LRL=L

All parameters shown after the filespec are optional and need not be used. If no parameters are specified, the LIST command will LIST the file in ASCII format, and the logical record length (LRL) of the file will be read from the directory.

The parameters shown may be entered in the same command line, such as
LIST TESTFILE:0 (HEX,REC=5,LRL=80,P).

If an extension is not used in the LISTing filespec, a default of /TXT will be used. If no file with the /TXT extension is found, the LIST will search for a file with an extension of all blanks.

Here are some examples of how LIST handles the "default" file extension of /TXT.

LIST TESTFILE:0

The system will first search drive 0 for a file named TESTFILE/TXT. If not found, it will then search for a file named TESTFILE.

LIST TESTFILE

The system will search all active drives for a file called TESTFILE/TXT, and LIST the first file named TESTFILE/TXT it encounters. If this file is not found, it will search all active drives for a file named TESTFILE, again LISTing the first TESTFILE it encounters.

LIST TESTFILE/SCR

The system will search all drives for a file called TESTFILE/SCR and LIST the first file named TESTFILE/SCR encountered. If the file is not found, the LIST command will not search for TESTFILE/TXT.

The parameters of the LIST command will determine the output format of the information in the LISTed file. Refer to the following section for a complete explanation and the proper use of these parameters. Note that the NUM and LINE parameters are for ASCII LISTings only and will be ignored if the HEX parameter is specified.

PARAMETER: NUM

This parameter will number the lines of the file as they are sent to the video display or printer. Line numbers will start with one (1) and be in the format 0001. Lines are determined by the occurrence of a carriage return. Linefeed characters will not generate a new line number.

PARAMETER: LINE

(This parameter may NOT be abbreviated). The LINE parameter is used with ASCII files. It will start the LISTing with the specified line of the file. Lines are determined by the occurrence of a carriage return character in the file. An example of the proper syntax would be LIST TESTFILE/TXT (LINE=14). This would LIST the ASCII file TESTFILE/TXT, starting with the line of the file after the 13th carriage return.

IMPORTANT: The NUM and LINE parameters will ALWAYS be ignored if the HEX parameter is specified.

PARAMETER: HEX

The HEX parameter will cause the file to be LISTed in the following format.

```
aaaa:bb = cc cc cc cc cc cc cc cc  cc cc cc cc cc cc cc cc
          d  d  d  d  d  d  d  d   d  d  d  d  d  d  d  d
```

aaaa represents the current logical record of the file in HEX format, starting with record 0.

bb represents the offset from the first byte of the current logical record (bb will be in hexadecimal notation).

cc will be the hexadecimal representation of the byte LISTED.

d will be the ASCII representation of the byte. A period (.) will be used for all non-displayable bytes.

For example, the command LIST LBASIC/CMD.BASIC (H) would produce a display as shown here:

```
0000:00 = 01 02 00 52 C3 06 5C 00  4E 45 58 54 20 77 69 74
           .  .  .  R  .  .  /  .   N  E  X  T     W  I  T
0000:10 = 68 6F 75 74 20 46 4F 52  00 53 79 6E 74 61 78 20
           H  O  U  T     F  O  R   .  S  Y  N  T  A  X
```

This is a LISTing of the file LBASIC/CMD.BASIC in HEX format. The logical record length was not specified in the command, and was found from the directory to be 256.

PARAMETER: REC

This parameter is used for LISTing HEX files. It tells the LIST command to start with the specified logical record number of the file. The first record in a file is record 0. When specifying a record number, REC=1 would LIST the second record of the file. The command LIST MONITOR/CMD (H,R=5) would start the LISTing with the sixth record. If this parameter is not specified, the LISTing will start with record 0.

PARAMETER: LRL

This parameter tells the LIST command to format the output using the specified LRL for each record. If the LRL parameter is not specified, the LIST command will use the record length in the file's directory entry. The LRL parameter is valid only when used with the HEX parameter.

PARAMETER: P

This parameter directs the output to the line printer rather than the video display. It may be used in conjunction with any of the other LIST parameters. Be sure the printer is enabled before using this command or the system will lock up.

PARAMETER: TAB

This parameter will cause the expansion of any TAB characters (X'09') encountered during ASCII LISTings to the video display or line printer. The tab locations are at columns 8, 16, 24 32, 40, 48, and 56.

The following examples will show some different LIST commands.

```
LIST MONITOR/CMD (HEX,LRL=8,REC=0)
LIST MONITOR/CMD (H,L=8)
```

These two commands will produce identical results - LISTing a file called MONITOR/CMD to the video display, using a LRL of 8, and starting with the first record of the file. The second example has merely substituted the abbreviations for the REX and LRL parameters, and let the RECord parameter default to 0. This LISTing display will be only 8 bytes wide, as the LRL is also the display width (for LRL's = 1 to 16). Maximum display width is 16 bytes per line. The same line width applies to LISTings sent to the printer with the (P) option.

```
LIST REPLY/PCL (NUM,TAB,P)
LIST REPLY/PCL (N,T,P)
```

These two commands produce identical results. The second example merely substitutes abbreviations for the parameters. The result of this command would be to send a LISTing of the file REPLY/PCL to the printer, using ASCII format, expanding all TAB characters encountered and NUMbering each new line that is printed.

```
LIST TESTFILE/OBJ (NUM,HEX,REC=5)
LIST TESTFILE/OBJ (HEX,REC=5)
```

These commands produce identical LISTings of the file TESTFILE/OBJ. Remember that the NUM and LINE parameters are ALWAYS superseded by the HEX parameter.

L O A D
=======

The LOAD command will load a load module format file (such as a /CMD or a /CIM) into memory without execution. The syntax is:

```
+-------------------------------------------------------------+
|  LOAD (X) filespec                                          |
|                                                             |
|  filespec is any valid LDOS filespec that is in load        |
|           module format.                                    |
|                                                             |
|  (x)      is an optional parameter for a LOAD from a        |
|           non-system diskette.                              |
|                                                             |
|  abbr: NONE                                                 |
|                                                             |
+-------------------------------------------------------------+
```

The LOAD command allows you to LOAD into memory a disk file that is in the proper format. The default file extension for the LOAD command is /CMD.

The following address restrictions exist when LOADing programs:

LOAD **Program must reside at or above X'5200'.**

LOAD (x) **Program must reside at or above X'5300'.**

After a program is LOADed, control is returned to the system without execution of the LOADed program.

The (x) parameter allows the LOADing of files that reside on a system or non-system disk. This is primarily useful for single drive owners, as a file may be loaded from a disk other than an LDOS system disk. The system will prompt you to insert the diskette with the LOAD file on it with the message:

INSERT SOURCE DISK <ENTER>

After the LOAD is complete, you will be prompted to place the system diskette back in drive 0 with the message:

INSERT SYSTEM DISK <ENTER>

At this point, the LOAD is complete.

M E M O R Y
==========

The MEMORY command allows you to reserve a portion of memory, see the current HIGH$ (highest byte of unused memory), modify a memory address, or jump to a specified memory location. The syntax is:

MEMORY (HIGH=addr,ADD=addr,WORD=dddd,BYTE=dd,GO=addr)	
MEMORY (CLEAR)	
CLEAR	Will fill memory from X'5200' to HIGH$ with X'00'.
HIGH=	Will set the specified address as HIGH$. addr must be less than the current HIGH$.
ADD=	Displays the word at the specified address. Also specifies the address for WORD and BYTE.
WORD=	Changes the contents of ADD and ADD+1.
BYTE=	Changes the contents of ADD.
GO=	Transfers control to the specified address. This parameter is always executed last.
addr	Any address in HEX or DECIMAL notation.
dddd	Any HEX "word" other than X'0000'.
dd	Any byte in HEX notation, other than X'FF'.
abbr: HIGH=H, ADD=A, WORD=W, BYTE=B	

In all MEMORY commands, the GO parameter, if used, will be the last parameter to be executed) regardless of its physical position in the command line. All other parameters will be acted upon before the actual GO is done.

The following restrictions are placed on the WORD and BYTE parameters:

WORD: Cannot = X'0000' or decimal value 0.
BYTE: Cannot = X'FF' or decimal value 255.

Refer to the following examples and descriptions.

MEMORY with no parameters will display HIGH$ (highest unused memory location) in the format X'nnnn'.

MEMORY (HIGH=X'E000')

This command would set HIGH$ to memory address X'E000', as long as the existing HIGH$ was above this location. The MEMORY command will only move HIGH$ lower in memory. The RESET command will allow you to RESET HIGH$ to the top of memory.

MEMORY (ADD=X'4049')

This command will display the contents of memory locations X'4049' and X'404A'. The display will be in the following format:

```
X'4049' = 16457 (X'00E0')  HIGH = X'E000'
  ----    -----    ----            ----
  aaaa    bbbbb    cccc            dddd
```

aaaa...is the ADDress specified in HEX notation.
bbbb...is the decimal equivalent of ADD.
cccc...is the contents of ADDress and ADDress+1, in LO-HI format.
dddd...is the current HIGH$ address.

MEMORY (ADD=X'E100',WORD=X'0A3E')

This command will modify memory locations ADD (X'E100') and ADD+1 (X'E101'), changing them to the value of WORD. They will be modified in standard LO-HI (reverse) format) with location X'E100' changing to X'3E' and X'E101' changing to X'0A'. The following would be displayed after this command:

```
X'E100' = 57600 (X'0000' => X'3E0A')  High = X'E000'
  ----    -----    ----       ----            ----
  aaaa    bbbbb    cccc       XXXX            dddd
```

All of the display is identical to the last example, except the contents of the WORD changed will be shown, represented in the display as XXXX.

MEMORY (ADD=X'E100',BYTE=X'0D')

This command will change the BYTE of memory at the specified ADDress (X'E100') to X'0D'. The display after executing this command would be:

```
X'E100' = 57600 (X'0000' => X'0D00')  High = X'E000'
  ----    -----    ----       ----            ----
  aaaa    bbbbb    cccc       XX              dddd
```

All of the display is identical to the last example, except for the modified BYTE change shown here with the XX.

MEMORY (GO=X'E000')

This command would transfer control to memory ADDress X'E000'.
Note that the Go parameter may NOT be abbreviated.

*** P U R G E**
==========

The PURGE command allows for "CONTROLLED" multiple KILLs of disk files. The syntax is:

PURGE :d
PURGE :d (QUERY=sw,MPW="aa",D="bb",I,S)
PURGE partspec w/wcc:d (QUERY=sw,MPW="aa",D="bb",I,S)
PURGE -partspec w/wcc:d (QUERY=sw,MPW="aa",D="bb",I,S)

:d is an optional drivespec, defaults to 0.

partspec and -partspec are as described in the LDOS glossary and under general information.

wcc Wild-Card Character <$> used as necessary for masking characters.

QUERY QUERY each file before PURGEing it.

MPW= The disk Master Password.

D= Allows specifying a range of MOD dates.

I Specifies INVisible files.

S Specifies SYStem files.

aa represents the disk Master Password.

sw represents the switch ON or OFF, default ON.

bb represents a date entry as follows:

"M1/D1/Y1-M2/D2/Y2" indicates all files with MOD dates between the two dates specified, inclusive.

"M1/D1/Y1"' will indicate all files with a MOD date equal to the date specified.

"-M1/D1/Y1" will indicate all files with MOD dates less than or equal to the date specified.

"M1/D1/Y1-" indicates all files with MOD dates greater than or equal to the date specified.

abbr: ON=Y, OFF=N, QUERY=Q

The PURGE command allows the user to do multiple KILLs of disk files without the need to specify the individual filespecs. The user will be prompted for the disk's Master Password if it is a password other than "PASSWORD" and not passed with the MPW parameter. The PURGE command allows several parameters to be set, providing for a specific group or groups of files to be PURGEd.

If the Q (Query) parameter is not specified, or if Q is specified without a switch, Q=Y is automatically assumed) and you will be asked before each file is PURGEd. The MOD flag and date will be shown for each file.

PURGE defaults to VISible files only. To include INVisible and SYStem files, the I and S switches must be specified in the command line.

The D switch allows you to choose a range of MOD dates to be use as criteria for the PURGE. Only those files meeting the date range will be used. Files without dates will never be shown if the D switch was specified.

NOTE: The files BOOT/SYS and DIR/SYS are not able to be PURGEd and will NEVER appear during execution of any PURGE command.

Following are some examples and explanations of the PURGE command.

PURGE :0 (MPW="SECRET")

> This command will purge all VISible files on drive 0, assuming that the Master Password of the disk is SECRET. The PURGE will show each file and wait until a Y (Yes, PURGE it) or a N (No, don't PURGE it) is entered. To abort the PURGE, press the <BREAK> key at this prompt. If the Master Password does not match the password of the disk, the PURGE will abort with an error message.

PURGE :1 (Q=N,I,S)

> This is a very POWERFUL and DANGEROUS command. It will PURGE all files) including SYStem files, from drive 1. If the disk's Master Password is other than PASSWORD, you will be prompted for it. Once the PURGE starts, it will remove ALL files from the disk, except BOOT/SYS and DIR/SYS. YOU WILL NOT BE ASKED BEFORE EACH FILE IS PURGED - IT WILL BE AUTOMATIC!! In other words, you will end up with a blank, formatted disk! This is a very convenient way to clean a disk, but be sure you don't actually need any file on the disk.

PURGE /BAS:1 (Q=N)

> This command will first ask for the Master Password of the disk in drive 1 (if it is not PASSWORD). Once entered, it will PURGE all VISible files with the extension /BAS from the disk. You will NOT be asked (Y/N) before each file is killed as the QUERY was specified as N, for No QUERY desired. You will, however, be able to stop the purge activity by pressing the <BREAK> key.

PURGE $$EX1:0 (I)

This command will PURGE all non-SYStem files whose filename has the characters EX1 as the third, fourth, and fifth characters of the filename. The wildcard character ($) masks the first two characters of the filename (the filename may be more than 5 characters in length, as trailing characters in the field are ignored). The file extension will have no effect on this PURGE command. You will be asked before any file is actually PURGED, as Q was not specified and defaulted to Q=Y. INVisible and VISible files will both be shown, as the I switch was used.

PURGE /$$S:2

This command will PURGE all VISible files on drive 2 whose file extension contains 3 characters and ends in the letter S. This would PURGE files with the extension of /BAS, for example. However, it would not PURGE the SYStem files, as the S switch was not specified. You will be prompted before each file is PURGEd.

PURGE -/CMD:0 (I)

This command will PURGE all non-SYStem files EXCEPT those whose extension is /CMD. You will be asked before each file is PURGEd.

PURGE :1 (D="02/01/81-02/04/81")

This command will PURGE all VISible files on drive 1, as long as their MOD date is between 02/01/81 and 02/04/81, inclusive. You will be asked before each file is PURGEd.

PURGE /SCR:2 (Q=N,D="-06/02/81")

This command will PURGE all VISible files with a /SCR extension, provided their MOD date is 06/02/81 or earlier. You will not be asked before each file is PURGEd.

R E N A M E
==========

This command will RENAME a file. The syntax is.

```
============================================================
|  RENAME filespec1 TO filespec2                           |
|  RENAME filespec TO partspec                             |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

The RENAME command allows you to change the filename and extension of a given file. The RENAME command will use dynamic defaults for the filename, extension, and drivespec of filespec2. This means that any part of filespec2 that is not specified will default to that of filespec1. The drivespec of filespec2, if specified, MUST be the same as that of filespec1 or the RENAME will abort and an error message will be generated.

RENAME will not allow the changing or deleting of a file's password. To change or alter a password, refer to the ATTRIB LIBrary command.

RENAME TEST/DAT:0 TO OLD/DAT

This command will RENAME the file TEST/DAT on drive 0 to OLD/DAT.

RENAME TEST/DAT:0 TO REAL

This command would RENAME the file TEST/DAT on drive 0 to REAL/DAT. The extension /DAT was not specified for filespec2, and defaulted to that of filespec1.

RENAME TEST/DAT:0 TO REAL/

This command will RENAME the file TEST/DAT on drive 0 to a file named REAL. The use of the / with no characters after it in filespec2 kept the extension from defaulting to /DAT.

RENAME TEST/DAT TO REAL/DAT

This command will search the active drives for the file TEST/DAT and RENAME it REAL/DAT.

RENAME TEST/DAT TO /OLD

This command will search the active drives for a file TEST/DAT and RENAME it TEST/OLD. The filename was not specified in filespec2, and defaulted to that of filespec1.

RENAME DATA/NEW.SECRET:1 TO /OLD

This command will RENAME the password protected file DATA/NEW.SECRET on drive 1 to DATA/OLD.SECRET. The filename and password for filespec2 defaulted to those of filespec1.

RENAME TEST/DAT TO TEST

This command is not a valid command, and will produce the error message DUPLICATE FILE NAME. This is because the extension of filespec2 will default to /DAT, thereby creating the same filename for filespec2 and filespec1, which are the same file.

R E S E T
=========

This command will RESET logical devices and provide a way to restore HIGH$ (highest unused memory location) to the top of memory.

```
+-------------------------------------------------------------+
|  RESET                                                      |
|  RESET devspec                                              |
|                                                             |
|  abbr: NONE                                                 |
|                                                             |
+-------------------------------------------------------------+
```

There are two uses for the RESET command. The first is a global RESET, the second is the RESET of a single device.

The global RESET will RESET all active devices, while the RESET of a single device will affect only that device.

The "single device" RESET will restore the device to its normal power up state. If high memory was used when this device was altered, it will not be reclaimed by the system. However, some routines can re-use the same memory allocation if they are enabled again after being disabled or RESET.

The following will re-use their original memory allocation if re-activated after being disabled. See the individual sections for exact command syntax and specifications.

SPOOL library command	KI/DVR, including the TYPE and JKL Options	
MiniDOS/FLT	PR/FLT	KSM/FLT

Here are some examples of the RESET *devspec command.

RESET *PR

If you had your printer (*PR) FILTERed with the PR/FLT routine, the RESET *PR command would restore the normal I/O path between the printer DCB and its power up driver.

RESET *DU

Suppose you had a dummy device *DU ROUTEd to a disk file TEST/TXT, and had your printer (*PR) LINKed to *DU. This configuration would cause all output to the *PR to also go to *DU, and into the disk file TEST/TXT. If you RESET *DU, the device table would show *DU = Nil, and the file TEST/TXT would be closed. However, *PR would still be LINKed to *DU. Since *DU = Nil, any output sent to the *PR would be ignored by *DU. The printer (*PR) would function normally. To clear the LINK, issue a RESET *PR command. *DU would continue to be shown in the device table until the system is reBOOTed, *DU is KILLed, or a global RESET is performed.

The RESET command with no devspec will do a global RESET. All system logical devices will be returned to their normal power-up state. All user logical devices will be removed from the device control table. Any FILTERing, LINKing, ROUTEing, or SETting will be cancelled. All open files will be closed. The Drive Code Table will be returned to a standard four drive, 5 1/4" configuration. Any software write protection will be cancelled. If your system drive has been set to some drive other than physical drive 0, be sure to insert a system disk into physical drive 0 before performing a global RESET.

The system will also attempt to set HIGH$ to the top of the available memory. If certain system functions that use the task processor are active (such as SPOOL or the Type Ahead), the RESET cannot restore HIGH$ to the top of memory. If this is the case, the following message will appear.

CAN'T RESET MEMORY, BACKGROUND TASK(S) EXIST

To reset HIGH$, you must turn OFF the particular functions or RESET the individual device before doing the RESET.

R O U T E
=========

The ROUTE command re-ROUTEs input/output for a specified logical device or creates a device. The syntax is:

```
============================================================
|  ROUTE devspec1 TO filespec1/devspec2                    |
|  ROUTE devspec (NIL)                                     |
|                                                          |
|  (NIL) is a bit-bucket.                                  |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

ROUTE will re-route all I/O for a specified logical device to another logical device, to a disk file, or (NIL). The (NIL), or bit-bucket, means that the device is ROUTEd to nothing. Any input sent to a device ROUTEd (NIL) will simply be ignored. A device ROUTEd (NIL) has no output.

NOTE: No more than 4 devices may be ROUTEd at any one time!

Anytime a device is ROUTEd to a filespec, a File Control Block (FCB) and a blocking buffer will be dynamically allocated in high memory. The system will determine the current HIGH$ (highest unused memory location) and use the space directly below this location for its buffer. HIGH$ will then be decremented to protect this area.

IT IS IMPORTANT TO NOTE: if the designated filespec already exists, the data ROUTEd to that file will be APPENDed to the end of the existing file. If you wish the data to be written from the beginning of the file, the file must be KILLed before the ROUTE is established.

A new logical device may be created with the ROUTE command. To create a device, simply ROUTE the desired devspec to another devspec, to a filespec, or to (NIL). The new device will then appear in the device table.

To examine any currently existing ROUTEing, use the DEVICE command. The device notations shown directly below the disk drive configurations will indicate all currently recognized devspecs and any ROUTEing, among other things, that has been done.

Once a device has been ROUTEd, it may be returned to its normal power-up state or removed completely from the device table with the RESET or KILL commands.

ROUTE *PR *DO

This command will ROUTE any data sent to the line printer (*PR) to the video display (*DO). None of the characters will be printed by the line printer, but instead will be shown on the video display. This command is very similar to LINK *PR *DO, the exception being that the characters are not printed by the line printer with the ROUTE but are with the LINK. The line printer need not be hooked to the system if *PR is ROUTEd to *DO. To remove the ROUTEing, use the command RESET *PR.

ROUTE *DU TO TEST/TXT:0

This command will ROUTE a user device (*DU) to a disk file TEST/TXT on drive 0. A File Control Block and a blocking buffer will be established in high memory. The device table will show the ROUTEing with an entry of:

*DU => TEST/TXT:0

The file TEST/TXT will remain open as long as the device *DU is not RESET. The file MUST be closed with the RESET *DU command prior to removing the diskette from the drive.

ROUTE *PR TO PRINTER/DAT

This command ROUTEs all data normally sent to the line printer (*PR) to a disk file PRINTER/DAT. The system will search all active drives and use the first file PRINTER/DAT it finds. Any data sent to the *PR will then be APPENDed to the end of the PRINTER/DAT file. If the file does not exist, it will be created on the first available drive. An FCB and blocking buffer will be allocated in high memory and the file PRINTER/DAT will remain open until the *PR is RESET.

Before ROUTEing any device to a disk file, it is advisable to determine the amount of FREE space available on the diskette. Make sure the space available on the disk is adequate to hold the amount of data you wish to ROUTE to it! A "Disk space full" error will hang up the system if encountered when writing to a file via the ROUTE command.

The constant "EOF maintenance" file mode may be useful to invoke when ROUTEing to disk files (see use of the "trailing ! character" with filespecs, in GENERAL INFORMATION and the Glossary). This will cause the EOF (End Of File) to be updated after each buffer is written to the file. If EOF maintenance is not invoked, then the EOF will not be written to the file until the "ROUTEing" is RESET, which will properly close the file. If a file is not properly closed, the data written to it may not be recoverable. If a "Disk space full" error is encountered when the EOF maintenance has been invoked, all data up to the last "full" buffer written to the file will be intact, and the file will be readable by the system.

R U N
=====

The RUN command will load a program into memory and then execute it. The program must be in load module format. The syntax is:

RUN (x) filespec (parm,parm,...)	
filespec	is any valid LDOS filespec of a file in load module format.
(x)	is optional to execute the program from a non-system disk for the single drive user.
parm	optional parameters to be passed to the filespec program.
abbr: NONE	

The RUN command will load in a load module format program that resides above X'51FF' and then execute the program. The default extension for the filespec is /CMD. If the program resides on a non-system diskette, the (x) parameter may be specified to load the program from that diskette and begin execution only after a system disk has been reinserted in drive 0.

****** NOTE ******

IF THE (x) PARAMETER IS USED THE PROGRAM MUST LOAD ABOVE X'52FF' .

Load module format programs may also be directly loaded and executed from the LDOS READY prompt by simply typing in the name of the program.

Following are some examples of the RUN command.

```
RUN SCRIPSIT/LC
SCRIPSIT/LC
```

Both of these commands will produce the same results. The program SCRIPSIT/LC will be loaded into memory and executed.

```
RUN BASIC
BASIC
```

Both of these commands will produce the same results. They will load a program named BASIC/CMD and execute it. Note that the file extension defaulted to /CMD when not specified by the RUN command.

RUN (x) INVADERS/CMD

This command is for the single drive user. It will LOAD the program INVADERS/CMD from any disk, whether or not it is an LDOS SYStem disk. After the command has been entered, you will be prompted with the message:

INSERT SOURCE DISK <ENTER>

At this point, you should insert the diskette containing the program in drive 0 and press <ENTER>. After the program is loaded, you will be prompted:

INSERT SYSTEM DISK (ENTER>

You should now insert your LDOS system disk back into drive 0 and press <ENTER>. Program execution will begin at this point.

The RUN command is identical to the LOAD command except for the fact that control is transferred to the program module "transfer address" rather than returning to the system.

S E T
=====

This command SETs a logical device to a DRIVER routine. The syntax is:

SET devspec TO filespec (parm,parm,...)
devspec any currently enabled logical device.
filespec any valid "driver type" program.
parms optional parameters required by the driver program specified with filespec.
abbr: NONE (except as allowed by the DRIVER program).

The SET command will SET a logical device to a driver program. It does this by LOADing the specified driver into high memory just below HIGH$. HIGH$ will then be decremented to protect this driver routine. Once a device is SET) any I/O to or from the device will be controlled by the driver routine. LDOS will allow the passing of parameters to the driver program. These parameters are totally independent of the SET command, and are determined only by the needs of the driver program.

When a device is SET, any previous FILTER, ROUTE, LINK, or SET of that device will be destroyed. Once a device has been SET, it will remain SET until it is either ROUTEd or RESET. The driver program will remain in high memory even if the device is RESET.

The KI/DVR program will re-use its original high memory allocation if RESET and then SET again. See the KI/DVR section for exact details.

All other SETting of devices will produce the following results:

If a device is RESET, and then SET again, the driver routine will load in below the current HIGH$. As a result, the SETting, RESETing, and then SETting again of devices will cause the available memory to continue shrinking. Once a driver program is loaded, it will not be removed from memory or overwritten, even if the same device is RESET and then SET to the same driver. A global RESET (if allowable) will remove this driver program and free up the memory by RESETting HIGH$.

The filespec parameter is the filespec of the driver program. The default extension for this file is /DVR.

For complete information on the Device Driver programs supplied with the LDOS system, refer to the section on DEVICE DRIVERS.

```
SET *CL TO RS232T/DVR (BAUD=300,WORD=7)
SET *CL RS232T (BAUD=300,WORD=7)
```

These two commands produce identical results. The TO is optional and may be replaced by a single space. Specifying the filespec RS232T/DVR produces the same results as specifying the filespec RS232T, as the default extension is /DVR.

These commands SET the Comm Line (*CL) to a driver routine called RS232T/DVR. This is an actual LDOS driver program, and is described in the DEVICE DRIVER section. The parameters BAUD and WORD are valid parameters of the RS232T/DVR program. All I/O to/from the Comm Line will be sent through this driver routine, and be properly dealt with to be sent out the RS-232 interface.

```
SET *KI INKEY (F=64)
```

This command SETs the keyboard (*KI) to a user driver program named INKEY/DVR, and passes the parameter F=64 to the driver program.

```
SET *PR SERIALPR
```

This command would SET *PR (the line printer) to a driver program named SERIALPR/DVR. The driver program might be a serial printer driver that would properly initialize the RS-232 port, allowing the use of a serial line printer.

S P O O L
=========

The SPOOL command establishes a FIFO (First In, First Out) buffer for a specified device (usually a line printer). The syntax is:

SPOOL devspec TO filespec (MEM=aa,DISK=bb)
SPOOL devspec (OFF)

devspec is any valid LDOS device.

filespec is an optional LDOS filespec.

OFF turns off the SPOOLer and RESETs devspec.

MEM= Memory to be used by the SPOOLer.

DISK= Disk space to be used by the SPOOLer (0 to bb)

aa Amount of memory to be used) in blocks of 1K (1024 bytes). 1K is automatically used.

bb Amount of disk space to be used in blocks of 1K (1024 bytes).

abbr: DISK=D, MEM=M, OFF=N

The SPOOL command will establish a FIFO buffer for a specified device. All output sent to the device will be placed in an output buffer consisting of memory and/or disk buffers, and will be sent to the device whenever that device is available to accept this data.

The minimum amount of memory required by the SPOOL command is 1K (1024 bytes) for the memory buffer. The filespec is optional, and if no disk buffers are required, it is possible to SPOOL strictly to memory.

When the SPOOLer is active, output to the specified device is treated in the following manner. Any output data which cannot immediately be accepted by the device is sent to the memory buffer. When the memory buffer is full, the data is sent to the disk buffer (if one has been specified). The stored information is sent to the device in a FIFO manner. Output of the stored data to the device is carried on as a background task even when the system is performing other functions.

The following rules govern the memory and disk space allocation:

PARAMETER: MEM

As stated earlier, the SPOOL command will ALWAYS require a minimum of 1K (1024) bytes for a memory buffer. If more memory is required, it may be allocated with this parameter.

```
MEM=10
```

This command will allocate 10K (10,240 bytes) of memory to be used as a SPOOL buffer. This memory will be dynamically allocated by the fully integrated SPOOL system processor to provide the most efficient operating environment, depending on the particular configuration you have established for your LDOS system.

If this parameter is not specified, 1K of memory will automatically be allocated for the SPOOL buffer.

PARAMETER: DISK

This parameter sets the maximum amount of file space to be allocated for the SPOOLing. Disk space is allocated in blocks of 1K (1024 bytes), the same as memory. When this parameter is set, the system will CREATE a file of the size specified. If this parameter is NOT specified, the SPOOL command will automatically allocate approximately 5K of disk space, depending on the particular disk type. The file name of this file will be determined by the filespec parameter.

To prevent the SPOOL command from using any disk space, specify this parameter:

```
(DISK=)
(DISK=0)
```

By specifying the DISK parameter with no size, the system will not allocate any disk space to the SPOOL command and will not CREATE any file.

PARAMETER: filespec

NOTE: For the filespec parameter to be valid, the DISK parameter must not have set the disk file allocation to zero.

The filespec is the name of the file the SPOOL command will write to any time its memory buffer is full. The default extension for this file is /SPL. The default filename will be the two letters of the devspec which is being SPOOLed. Refer to the following examples.

SPOOL *PR TEXTFILE:0...The filespec will be TEXTFILE/SPL:0, as the file extension was not specified and defaulted to /SPL.

SPOOL *PR /TXT:0...The filespec will be PR/TXT:0, as the filename was not specified and defaulted to the two letters of the devspec (the letters P and R, from the devspec *PR).

SPOOL *PR :1...This command will look on drive 1 for a file named PR/SPL. If the file PR/SPL:1 is not found, it will be CREATED on drive 1 with a length determined by the DISK= parameter.

SPOOL *PR.. This command will search all active drives for a file named PR/SPL. If this file is not found, the file PR/SPL will be CREATEd on the first available drive (with the file size determined from the DISK parameter).

The following examples will show some possible combinations of the allowable SPOOL parameters.

SPOOL *PR TEXTFILE:0 (MEM=5,DISK=15)

This command will allocate 5K of memory and 15K of disk space in a file named TEXTFILE/SPL on drive 0. Any output sent to the printer will be buffered and sent to the line printer (*PR) as fast as the printer can accept the characters. Even if current program printing functions exist) other functions will be carried out and the line printer will continue to receive data from the SPOOLed buffers as fast as it can accept the data. The other program function processing will be carried on with little noticeable interruption. If the 5K memory buffer is filled, the data will then be written to the disk file TEXTFILE/SPL on drive 0.

SPOOL *PR (MEM=10,DISK=)

This command will create a 10K memory buffer for any data that is to be sent to the line printer (*PR). If a printer command is received, the data will be immediately sent to the 10K memory buffer, and then SPOOLed to the line printer whenever the printer can accept it (i.e. whenever the printer is not printing or otherwise in a BUSY or FAULT state). Since the data is sent to the printer as a background task, normal program execution will continue to take place. Note that none of the SPOOLed data will be sent to a disk file, as the parameter DISK was specified without any size. If the memory buffer is filled, processing of the current program functions will halt until the line printer has printed enough data to bring the outstanding character count below 10K (the size of the memory buffer).

If you are running an applications program that involves output to the line printer, it is possible that the overall efficiency of the program may be improved by activating the LDOS SPOOLer. The size of the program and the available free memory and disk space will determine the amount of SPOOLing available for your needs.

ONE NOTE OF CAUTION: the SPOOL file on disk will remain open as long as the SPOOLer is active. DO NOT KILL this file or remove the diskette without first closing the file! Note also that if a SYSGEN is done, the file will remain open even after rebooting. This is not advisable unless the file is on drive 0. The file may be closed by turning the SPOOLed device OFF. The proper syntax is:

SPOOL devspec (OFF)
SPOOL devspec (N)

Either of these two commands will turn off the SPOOLer and close the associated disk file. Please note that the disk file will not be closed by RESETing or KILLing the SPOOLed device.

Once the SPOOLer is turned off, it may be turned on again. Doing so will re-use the same memory locations allocated when it was originally turned on. The following restrictions will apply:

The original parameters will be stored. If turned off and then back on, any parameters specified may not exceed the memory or disk parameters originally given, or an error will occur. However, memory or disk space parameters may be diminished.

The original stored parameters will not be affected by turning the SPOOLed device off and then back on.

*** S Y S T E M**
==========

This command is used to configure the user definable areas of your LDOS system. The syntax is:

```
+-------------------------------------------------------------+
|  SYSTEM (parm,parm,...)                                     |
|                                                             |
|  Allowable SYSTEM parameters are:                           |
|                                                             |
|          ALIVE          BASIC2          BLINK              |
|          BREAK          DRIVE           FAST               |
|          GRAPHIC        SLOW            SVC                |
|          SYSGEN         SYSRES          SYSTEM             |
|          TYPE                                              |
|                                                             |
|  Parameter arguments will be detailed in this section.      |
|                                                             |
|  abbr: ON=Y, OFF=N                                          |
|                                                             |
+-------------------------------------------------------------+
```

The existing configuration of your LDOS system can be seen by doing the DEVICE and MEMORY commands. The SYSTEM command can set or change the disk drive configuration as well as turn on or off different keyboard, video, and hardware drivers. Each valid SYSTEM command will be discussed in this section.

Once your LDOS system has been configured, you may store the configuration on the drive 0 disk with the SYSTEM (SYSGEN) parameter. Please read this section thoroughly to determine the different SYSTEM command uses, and to discover exactly how other LDOS commands will affect the (SYSGEN) parameter.

Certain of the SYSTEM commands must load driver routines into high memory to accomplish their functions. When they do this, they determine the highest unprotected memory location (referred to as HIGH$) and load directly below this location. After loading, the LDOS system moves HIGH$ down to protect these routines. If you have executed any SYSTEM commands that require the use of this high memory, be aware that your overall free memory will be decreased accordingly.

Most of the following parameters for the SYSTEM command may be used together in the same command line. To do this observe the syntax: SYSTEM (parm,parm,....,parm). Each parameter must be accompanied by its switch or setting as required. Certain SYSTEM parameters will cause an exit from the SYSTEM command. Please be aware that SYSTEM parameters will be executed in the following order, regardless of their physical location in the command line.

1)SLOW	2) FAST	3) BASIC2	4) BREAK
5)BLINK	6) LARGE	7) SMALL	8) TYPE

9) GRAPHIC	10) ALIVE	11) SVC	12) SYSRES
13) DRIVE	14) WP	15) DISABLE	16) ENABLE
17) STEP	18) DELAY	19) DRIVER	20) SYSTEM
21) SYSGEN			

Following are complete descriptions of the SYSTEM parameters.

SYSTEM (FAST)
SYSTEM (SLOW)

These commands are used only if a suitable clock speed-up modification has been installed in the TRS-80 computer unit. (Speed up kits are neither endorsed or condemned by L.S.I., although some effort has been made to support this type of modification). They will modify certain timing loops in the LDOS system to accommodate the current processor clock speed, as well as switching the software controlled clock. These two commands have precedence over any other SYSTEM command, and will always be executed before any of the other commands are carried out. No memory will be used by these parameters.

The clock speed is controlled in the following manner:

FAST issues an OUT Port 254,1 command.

SLOW issues an OUT Port 254,0 command.

SYSTEM (ALIVE=switch)

The SYSTEM (ALIVE) parameter displays an "ALIVE" character in the upper right corner of the screen. It is primarily useful to determine the current state of the task processor. If the ALIVE bug is moving, the task processor is running. Note that the ALIVE bug may continue moving (indicating an ALIVE system) even when the TRACE display has stopped.

The switch is either ON or OFF. If not specified, ON is assumed. The ALIVE parameter uses some RAM in high memory.

SYSTEM (BASIC2)

This command will direct you to the ROM Basic in the TRS-80 computer.

Typing in SYSTEM (BASIC2) while in the LDOS READY mode will clear the screen, and "Cass ?" will be displayed in the upper left corner of the display. Any ROUTEing, LINKing, or DRIVER routines SET under LDOS will be reset to the normal ROM Basic drivers. While in ROM BASIC, none of the disk functions are available for use and you cannot return directly to LDOS or LBASIC. You must press the reset button or turn off the computer and go through power up to get back to the operating system.

SYSTEM (BREAK=switch)

This command will enable or disable the <BREAK> key. The allowable switches are ON or OFF. If switch is not specified, the default will be ON. Once the <BREAK> key is disabled by doing a SYSTEM (BREAK=OFF) command, pressing it will have no effect, and the system break bit will not be set. It may be re-enabled at any time by doing a SYSTEM (BREAK=ON) command. The (BREAK=ON) will also enable the <BREAK> key if it was disabled by the AUTO LIBrary command. No memory will be used with this parameter.

NOTE: Specifying (BREAK=OFF) will prevent routines such as the BUILD Library command from exiting when the <BREAK> key is pressed!

SYSTEM (BLINK=aaaa)

This command controls the LDOS cursor character. The parameter aaaa can be represented as ON/OFF or as a decimal value. The cursor character numbers in the following examples are the ASCII values (in decimal) of the TRS-80 character set. This command does not use high memory.

ON.....Turns the blinking cursor on, with the cursor character being a graphics character (character 176).

OFF....Turns off the blinking cursor, leaving the cursor character unchanged.

aaaa can also be represented as any displayable ASCII character value. For example, if the command SYSTEM (BLINK=42) were given, the blinking cursor character would be an asterisk (character 42).

SYSTEM (BLINK,LARGE)

This command turns on a large (character 143) blinking cursor.

SYSTEM (BLINK,SMALL)

This command turns on a small (character 136) blinking cursor.

SYSTEM (DRIVE=d,parm,parm,...)

This command sets certain parameters for the disk drives in your system. Refer to the following for explanation of the allowable parameters. This command uses no extra memory. Please note that the drive type (5 1/4'', 8'', or HARD DISK) and the location of a drive are set by the SYSTEM (DRIVE=, DRIVER) command.

DRIVE=d (Where d is any valid drive number in your system.)

The parameters can be any of the following:

DELAY=OFF....This command is valid only for 5 1/4'' drives. The DELAY is the time allowed between drive motor start up and the first attempted read of the diskette in that drive. The OFF parameter sets this delay to .5 seconds.

DELAY=ON.....This command is valid only for 5 1/4'' drives. The ON parameter sets the delay between drive motor start up and the first attempted read to 1 second. This is the normal DELAY time for all 5 1/4'' drives.

DISABLE......This command will remove the specified drive number from the Drive Code table. Once disabled, any attempt to access that drive will cause the message "Illegal Drive Number" to appear. The drive can be re-enabled with the ENABLE parameter.

ENABLE.......This command will enable the specified drive number and place its configuration information in the Drive Code table. If you enable a drive that has not been set up with the SYSTEM (DRIVE=,DRIVER) command, totally unpredictable results may follow.

STEP=n.......This parameter will set the stepping rate for the specified drive number, where n is a number 0 to 3. The following table lists the different stepping rates for 8'' and 5 1/4'' drives.

```
n=0     8'' step rate = 3  ms    5 1/4'' step rate = 6  ms
n=1     8'' step rate = 6  ms    5 1/4'' step rate = 12 ms
n=2     8'' step rate = 10 ms    5 1/4'' step rate = 20 ms
n=3     8'' step rate = 15 ms    5 1/4'' step rate = 30 ms
```

SYSTEM (DRIVE=d,DRIVER="filespec")

To access the disk drives, LDOS will use information stored in memory in the Drive Code Table (DCT). No special configuration should have to be done unless drives other than 5 1/4" floppy drives are used. To configure the system for other drive types, it will be necessary to use the SYSTEM (DRIVER) command.

If the command is entered without the filespec parameter, you will be prompted for the DCT program as follows:

```
ENTER DCT DRIVER <CR DEFAULT>
```

At this point you could enter the filespec of your disk driver program. Note that the default file extension is /DCT.

LDOS will come with four 5 1/4" drives as the standard configuration. Upon initial power up, the standard drive configuration will appear as follows:

```
CYL=40, DDEN, SIDES=1, STEP=30, DELAY=1
```

The purpose of the DRIVER command is to parse the command line parameters and then call a disk driver program that will place the correct information about a disk drive type in the DCT. Please note that the file extension /DCT is the default extension for disk driver programs.

Information about specific use of this command will be provided with the driver program for the particular hardware.

SYSTEM (DRIVE=d,WP=sw)

This command will allow you to software write protect any or all drives currently enabled. The parameters are as follows:

d = the drive number affected.

sw = the switch ON or OFF. ON will set the write protect status, and OFF will remove it and allow the drive to be written to.

The command with no drivespec specified will act globally. That is, SYSTEM (WP=ON) will write protect all drives in the system, and SYSTEM (WP=OFF) will remove any software write protection that has been done on any drive. The WP=OFF parameter will have no effect on a disk physically protected with a write protect tab.

Note that if the flag ON or OFF is not specified, ON is assumed.

SYSTEM (GRAPHIC)

This command informs the LDOS system that your lineprinter has the capability to reproduce the TRS-80 graphics characters during a screen print. If this parameter is used, any graphics characters on the screen will be sent to the lineprinter during a screen print command, either from the DOS level or with LBASIC's CMD"*". DO NOT use this parameter unless your printer is capable of reproducing the TRS-80 graphics characters.

SYSTEM (TYPE=switch)

This command will turn on or off the task processing of the KI/DVR type ahead feature. If you have SET *KI to the KI/DVR driver and wish to temporarily suspend the type ahead feature, use the SYSTEM (TYPE=OFF) command. The type ahead task processing may be restarted with the SYSTEM (TYPE=ON) command.

SYSTEM (SVC)

This command will load a SuperVisory Call (SVC) table into high memory. A complete description of the SVC table can be found in the technical section of the manual.

SYSTEM (SYSGEN=switch)

This command creates or deletes a configuration file on the drive 0 diskette, where switch represents the following parameters:

ON.....creates a configuration file.

OFF....deletes the configuration file.

If switch is not specified, ON is assumed - SYSTEM (SYSGEN) is the same as SYSTEM (SYSGEN=ON). After a SYSTEM (SYSGEN=OFF) command has been given, the current configuration of the system will not change until the system is booted again.

When the SYSGEN parameter is used, all current device and driver configurations will be stored on the diskette in drive 0. A file named CONFIG/SYS will be created to hold the configuration. Each time the system is booted, the configuration stored in this file will be loaded and set. To prevent this automatic configuration, hold down the <CLEAR> key while the boot is in progress. Note that the system configuration will take place before any AUTOed command is executed. In addition to the SYSTEM commands and parameters listed in this section, the following will be stored in the configuration file by a SYSTEM (SYSGEN) command:

1) All FILTERing, LINKing, ROUTEing, or SETting that has been done.

2) Any active background tasks (TRACE, SPOOL, CLOCK, or DEBUG).

3) Any special utility routines loaded into high memory and protected with the MEMORY command. All memory from HIGH$ to the physical top of memory will be written to the CONFIG/SYS file.

4) The present state of VERIFY (ON/OFF) and of the keyboard caps lock (the <SHIFT><0>).

SYSTEM (SYSRES=n)

This command will allow you to reside certain LDOS SYStem overlays in high memory. SYS files 1-5, and 8-11 may be loaded using this command. Note that each overlay will require 1K (1024 bytes) of memory.

Having certain of these SYS overlays resident in memory will speed up most disk I/O operations, as these modules will not have to be loaded from disk. It will also allow you to purge these overlays from your system disk, providing more room for data and programs. Overlays 2, 3, 8, and 10 must be resident for certain types of BACKUPs (see the BACKUP Utility section).

The DEVICE library command will show any overlays that are currently resident in high memory.

SYSTEM (SYSTEM=n)

This command will allow you to assign a drive other than drive 0 as your system drive. It will do this by swapping the DCT information of the drive specified with the current system drive. Note that there must be a system disk in the drive specified!

Once this command has executed, LDOS will look for any needed SYStem files on the new system drive. The logical drive numbers will also be changed - addressing the original system drive will now access the newly specified system drive) and vice versa.

This procedure may be repeated, and a swap of the current system drive with the drive specified will occur. The logical drive numbers will also change again. Be careful when repeating this command, or you may lose track of which drive is currently assigned to what logical drive number.

Note that doing a global RESET Library command will reset all drive DCTs to the default 5 1/4'' configuration. Be sure to have a system disk in physical drive 0 before performing a global RESET command.

*** T I M E**
=========

This command is used to set the TIME for the "real time" clock.

TIME hh:mm:ss
TIME
hh = hours 00-23
: = mandatory colon
mm = minutes 00-59
: = mandatory colon
ss = seconds 00-59
hours, minutes, and seconds must consist of two numbers each.
abbr: NONE

After entering the following commands:

CLOCK <ENTER>
TIME 01:20:22 <ENTER>

The TIME displayed on the clock in the upper right corner of the screen will start at 01:20:22. That's one hour, 20 minutes and 22 seconds after the hour of midnight. It will increment in seconds after the point where you press <ENTER>.

Note that the time lag between pressing the <ENTER> key and the time that will actually be set on the real time clock will be about 2 seconds (the time needed to execute the TIME command). It is usually best to type in the command TIME and then the time plus several seconds after the correct time. Wait for (seconds -2) to come up on your watch and press <ENTER>. This will give you the correct TIME on the "real time" clock. The TIME command is used to set the internal time clock, whether the CLOCK display function is used or not this internal "time piece" is running. To view the TIME that the "real time" clock is keeping, use the CLOCK command.

Entering the command TIME with no parameters will display the current setting of the real time clock.

*** T R A C E**
==========

This command displays the user's Program Counter address (PC register of the Z-80 processor) in the upper right corner of the video display. The syntax is:

```
============================================================
|  TRACE                                                   |
|  TRACE (ON)                                              |
|  TRACE (OFF)                                             |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

This command will display the contents of the Program Counter on the video display. The display will be in HEXadecimal format. Any information normally displayed on the top line (print locations 45 - 48) will be overwritten by the TRACE display. The display is constantly updated as a background task. The TRACE command is primarily useful during execution of assembly language programs.

The allowable commands are:

TRACE (ON) Turns the TRACE on.

TRACE (OFF) Turns the TRACE off.

TRACE Turns the TRACE on, the default parameter being ON.

NOTE: TRACE will not function properly when the display is in the 32 character mode.

*** V E R I F Y**
==========

The VERIFY command forces all disk writes to be VERIFYed with a read-after-write operation. The syntax is:

VERIFY (flag)
flag is the parameter ON or OFF
abbr: NONE

The VERIFY command will determine whether or not writes to a disk file are VERIFYed with a read-after-write operation. The normal power up condition is VERIFY (OFF). To cause a VERIFY of every write operation, you must specify the command:

VERIFY (ON)

Although having the VERIFY function turned ON will provide the greatest reliability during disk I/O, it will also increase the overall processing time anytime a disk file is written to. The user must determine if the increase in reliability warrants the increase in processing time.

VERIFY (OFF) will disable the read-after-write VERIFY function.

Note that all disk writes will automatically be VERIFYed during BACKUP functions. Also, certain critical writes to system tables will always be VERIFYed.

B A C K U P
==========

The BACKUP utility is provided to duplicate data from a source disk to a destination disk. The syntax is:

BACKUP :s TO :d (parm,parm,...) <as shown below>
BACKUP partspec w/wcc:s TO :d (parm,parm,...)
BACKUP -partspec w/wcc:s TO :d (parm,parm,...)

:s	the SOURCE drivespec.
:d	the DESTINATION drivespec.
MPW="aa"	passes the source disk's Master Password.
SYS	indicates SYStem files.
INV	indicates INVisible files.
VIS	indicates VISible files.
MOD	indicates files MODified since last BACKUP.
QUERY	parameter indicating Query each file before moving. The switch ON or OFF may be specified.
OLD	will BACKUP only those files already existing on the destination disk.
NEW	will BACKUP only those files NOT already on the destination disk.
D=	will allow you to specify a range of MOD dates "M1/D1/Y1-M2/D2/Y2" will BACKUP only those files whose MOD dates fall between the two dates specified, inclusive. "M1/D1/Y1" will BACKUP all files with MOD dates equal to the specified date. "-M1/D1/Y1" will BACKUP all files with MOD dates less than or equal to the specified date. "M1/D1/Y1-" will BACKUP all files with MOD dates greater or equal to the specified date.
X	allows BACKUPs with no SYStem disk in drive 0.

abbr: QUERY=Q, ON=Y, OFF=N

****** EXTREMELY IMPORTANT ******

Due to the complexities involved with handling many different disk drive configurations, the LDOS BACKUP utility demands that destination disks MUST be FORMATted before the BACKUP begins. This FORMAT before BACKUP requirement is found in most large operating systems including the TRS-80 Model II. Having the destination disk FORMATted will allow the BACKUP utility to determine if a Mirror Image BACKUP is possible, or if it will be necessary for the BACKUP utility to do a Reconstruct.

The BACKUP command will move all or part of the data from a specified source disk to a specified destination disk. The parameters of the BACKUP command may be used to determine which data will be moved. All of the parameters are optional, with only the source and destination drivespecs being prompted for if not entered. If the source disk contains a password other than PASSWORD, it will be prompted for if not passed with the MPW= parameter.

BACKUP will remove the MOD flags from any source drive files that contain them. If the source disk is write protected, the following message will appear.

CAN'T CLEAR MOD FLAGS. SOURCE DISK IS WRITE PROTECTED.

Be aware that the MOD flags will not be removed if this is the case.

The BACKUP command utilizes three types of BACKUPs. They are MIRROR IMAGE BACKUP, BACKUP BY CLASS, and BACKUP RECONSTRUCT. The type of BACKUP will be determined in part from the drive information contained in the system's Drive Code Table. The following rules will determine the type of BACKUP that will take place:

1) Mirror Image BACKUPs may only be done between drives with matching configurations of SIDES, DENSITY, and drive type (5 1/4", 8", or hard).

2) Specifying any parameters except X or MPW, or specifying a filespec, partspec, or wcc will cause a BACKUP By Class to occur.

3) Backups between drives of different configurations will cause a BACKUP Reconstruct to be invoked, if a BACKUP By Class was not specified.

4) BACKUP By Class and BACKUP Reconstruct function identically.

The BACKUP command will function two separate ways in regard to the destination disk. Be sure to understand the difference between these two functions.

1) A Mirror Image BACKUP will do a cylinder-for-cylinder copy from the source to the destination disk. All cylinders including the DIRectory and the BOOT cylinder will be moved. When this BACKUP is finished, the destination disk will be an exact duplicate of the source disk, including the PACK ID.

2) A BACKUP By Class or a BACKUP Reconstruct is actually a mass copy function. The data to be moved will be copied one file at a time from the source to the destination disk. Files existing on the destination disk but not on the source disk will remain untouched by these BACKUPs. When these BACKUPs are finished, the destination disk will contain all files moved from the source disk PLUS any other files that existed on the destination disk before the BACKUP. The DIRectory and BOOT files are NOT copied from the source to the destination disk, and the destination disk PACK ID will remain the same.

If the drives in a Mirror Image BACKUP are not configured the same, a BACKUP Reconstruct will be invoked. If a Mirror Image BACKUP is desired, use the DEVICE command to log on diskettes of the proper configuration in the drives to be used.

All diskettes in the LDOS system will have a PACK ID assigned to them when they are FORMATted. This PACK ID consists of the disk Name and Master Password. During a Mirror Image BACKUP, the PACK ID's will be compared. If they are not the same, you will be asked whether or not to continue the BACKUP. Your LDOS Master Disk comes with the disk Name LDOS-5.1 and the disk Master Password of PASSWORD. The source disk PACK ID will be duplicated on the destination disk during a Mirror Image BACKUP.

If the disks in a Mirror Image BACKUP contain different Pack I.D.'s (disk Name and Master Password), the message:

DIFFERENT PACK ID'S! ABORT BACKUP?

will appear. Answer this prompt Y (yes, abort the BACKUP) or N (no, don't abort - please continue the BACKUP).

If the BACKUP utility is being accessed from a DO file, the PACK ID's must be the same. If they are not, the DO will abort when it reaches the DIFFERENT PACK ID'S question!! BACKUP (x) and single drive BACKUP are also invalid from a DO file.

The following question may appear during a Mirror Image BACKUP attempt:

CYLINDER COUNTS DIFFER - ATTEMPT MIRROR IMAGE BACKUP?

This message will appear if the destination and source disks have the same size, density, and number of sides but have a different number of cylinders.

Answer this prompt Y (yes, attempt Mirror Image) or N (no, not a Mirror Image BACKUP). If a mirror image BACKUP is done using disks with a different number of cylinders, the destination disk directory will be placed on the same cylinder as found on the source disk.

BACKUPs between drives of different types (i.e. 5'', 8'', and hard disk) are allowed, but may necessitate more than one destination disk if the data to be moved from the source disk exceeds the capacity of the destination disk. Be sure to have extra FORMATted diskettes ready if this is the case. The BACKUP command will prompt you with the following message if the destination disk is full.

DISK IS FULL. ENTER NEW FORMATTED DESTINATION DISK <ENTER>

At this point, insert the next FORMATted disk in the destination drive. This procedure may be repeated as many times as necessary to complete the BACKUP.

The X parameter will allow you to make a BACKUP without a SYStem disk in drive 0. There are two types of X BACKUPs - Mirror Image and By Class. If the BACKUP will be by Class or a Reconstruct, SYS overlays 2, 3, 8, and 10 must be resident in memory (see the SYSTEM (SYSRES=) library command). A Mirror Image will not require these overlays to be resident.

Following are some examples and descriptions of the BACKUP command. Please note that in all examples, the source disk's Master Password will be asked for if it is other than PASSWORD and is not specified with the MPW parameter. If the Q parameter is specified, the file's MOD date and MOD flag will be shown along with the filespec.

BACKUP :0 :1

This command will attempt a Mirror Image BACKUP, using drive 0 as the source drive and drive 1 as the destination drive. If the drives are differently configured, a BACKUP reconstruct will be invoked. All files will be moved from drive 0 to drive 1, with the exception of DIR/SYS and BOOT/SYS if a Reconstruct is invoked.

BACKUP $:0 :1

This command will force a BACKUP BY Class. The first prompt will be for the Master Password, if other than PASSWORD. The wcc (WildCard Character) causes a BACKUP by class. All files will be examined, and ALL files (except BOOT and DIR) will be copied because they will "match" the single wcc. This BACKUP is the way to force a reconstruct in situations where a Mirror Image would normally have been done. One reason for this might be to remove unwanted "extents" from disk files on the source drive by copying them onto a cleanly FORMATted destination drive.

BACKUP :0 :1 (Q)

This command will function identically to the previous example, except that you will be asked before each file is moved. You will also see the MOD date and flag.

BACKUP :1 :2 (VIS)

This command will copy all VISible files in drive 1's DIRectory to drive 2. A BACKUP By Class will automatically be invoked.

BACKUP :2 :1 (INV)

This command will copy all files that are INVisible in drive 2's DIRectory to drive 1, invoking a BACKUP By Class. Note that the SYStem files will not be copied.

BACKUP :0 :1 (SYS)

This command will BACKUP all SYStem files from drive 0 to drive 1, invoking a BACKUP By Class.

BACKUP :0 :1 (VIS,INV)

This command will BACKUP every VISible and INVisible user file from drive 0 to drive 1, invoking a BACKUP By Class. In other words, this command will copy all files except the SYStem files.

BACKUP :0 :1 (MOD,Q,MPW="SECRET")

This command will copy all files that have been MODified (written to) from drive 0 to drive 1. It will Query each file before it is copied, also showing the file's MOD date and flag. The Master Password was passed with the (MPW=) parameter and will not be asked for.

BACKUP /CMD:0 :1
BACKUP $/CMD:0 :1

This command will force a BACKUP By Class, with the file class specified as /CMD. All files with the extension /CMD will be copied from drive 0 to drive 1. Note that the wcc ($) has no actual effect on the BACKUP. Specifying the /CMD will look at ALL /CMD files, just as the $/CMD will. If the file exists on drive 1 it will be overwritten, otherwise it will be created at this time. No files on drive 1 will be touched except for the /CMD files copied from drive 0.

BACKUP $$$$$$AT:2 :3 (MOD)

This command will BACKUP all files whose filename is 8 characters long and contains AT as the last two letters. Only those files that meet this criteria and have been MODified will be copied. A BACKUP By Class will be invoked.

BACKUP /$$S:1 :2

This command will BACKUP all files whose extensions are 3 characters long, ending with the letter S. The wcc ($) masks the first two characters of the extension, so the extensions /BAS, /TSS, /SYS, etc. would all match. A BACKUP By Class will be invoked.

BACKUP -/CMD:0 :1

This command will BACKUP all files from drive 0 to drive 1, EXCLUDING those that have the extension /CMD.

BACKUP :1 :1

This command will BACKUP between two disks in drive 1. You will be prompted to switch between the source disk and destination disk at the appropriate times. Neither disk in drive 1 need be a SYStem disk. This command could be used to BACKUP a data disk. See the next example with the (x) parameter for another example of data disk BACKUPs.

BACKUP :0 :1 (x)

This command will BACKUP a disk in drive 0 to a disk in drive 1. Its primary use is to BACKUP non-SYStem diskettes, such as data diskettes. When using this BACKUP parameter, you will be prompted to insert the proper disk in drive 0. You may be prompted to re-insert a SYSTEM disk into drive 0 during certain BACKUPs. When the BACKUP is complete, you will be prompted to insert a SYStem disk back in drive 0. This command is useful for BACKUPs of data disks in a two drive system. If the BACKUP will be by Class or a Reconstruct, SYS overlays 2, 3, 8, and 10 must be resident in memory.

BACKUP :1 :2 (OLD)

This command will BACKUP all files from drive 1 to drive 2, only if they already exist on drive 2.

BACKUP :1 :2 (NEW,Q)

This command will BACKUP files from drive 1 to drive 2, only if they DO NOT already exist on drive 2. You will be prompted before each file is moved, as the Q parameter was specified.

BACKUP /ASM:3 :2 (D="05/06/81-05/10/81")

This command will BACKUP all files with the extension /ASM, as long as their MOD dates fell between the two dates specified, inclusive.

C O M M A N D F I L E (CMDFILE)

The LDOS COMMAND FILE Utility is a general purpose disk-to-disk, tape-to-disk and disk-to-tape) machine language program that has been designed to provide the capability of appending two or more CoMmanD (CMD, CIM, OBJ) files (machine language load modules) or SYSTEM tape files (machine language) files that can be loaded with the BASIC "SYSTEM" command. Inherent in its capability of performing I/O to disk or tape are the following functions:

1. Append two or more 'COMMAND' disk files or 'SYSTEM' cassette tape files into one file. This is useful to concatenate two or more separately assembled OBJECT code files, concatenate two or more non-contiguous blocks of code, or also couple two or more programs together so they load together.

2. Offset a tape or disk file so that it loads into a region other than originally programmed. A driver routine is optionally appended that moves it back to its original load region. User options provide for disabling the clock interrupts and keyboard debounce routines in the event that the SYSTEM program would have overlayed the debounce routine of TRSDOS or LDOS.

3. Machine language programs (tape or disk files) can be appended with patched code to correct errors in a manner similar to the PATCH/CMD. This operation requires use of an Editor Assembler and, of course, knowledge of the corrective patch code.

4. Command 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.

5. SYSTEM cassette tape files can be created from non-contiguous blocks of memory; heretofore only possible via direct assembly from the Editor Assembler.

6. For the disk user, during input of 'COMMAND' files, the load address range of each block of code is displayed to the CRT and optionally to a line printer. The file's transfer address or entry point is also displayed.

   TO ENTER THE COMMAND FILE UTILITY

   At LDOS READY simply type: CMDFILE <ENTER>

   COMMAND FILE will load and execute.

COMMAND STRUCTURE

All functions and procedures are specified by responding to a series of queries. Some queries request yes/no responses (abbreviated Y/N), some request disk/tape responses (abbreviated D/T), while others request specific information (i.e. file names, new addresses, etc.). Most yes/no and disk/tape responses can also be answered with a "C" to cancel the request and return to the main prompt as noted above. If you want to return to LDOS, responding with "E" for EXIT will return you to the respective system. Each query displays the valid responses acceptable to it. All queries accept lower case responses as well as upper case.

Query (1):

```
ADDRESS LOAD LOG TO PRINTER (Y,N,E)? >
```

The address load log will be displayed only for files read in from disk. The timing on tape reads is too critical to perform the extra processing necessary to detect the load limits and display them during a tape read. If you are a disk user, have a line printer, and want this log displayed on your printer, respond with a "Y", otherwise respond with an "N". If you want to exit CMDFILE, enter "E". This query is referred to as the main query. Whenever it is displayed, the memory buffer, used to store input files, will be reset to its beginning position to initialize for a series of input requests. This effectively "clears" the input buffer.

Query (2):

```
INPUT FROM DISK OR TAPE (D,T,E,C,Q) OR <ENTER> TO END READS? >
```

This query cycles anytime CMDFILE is ready to read in another file. Any file read in will be appended to any file previously input since the main query prompt. If you want to read in a disk file, respond with a "D". If the file is to be input from tape, respond with a "T". You may quit and return to LDOS by entering an "E". A response of "C" will cancel the input and return you to the main query, thus reinitializing the memory buffer. The <Q> response permits display of a diskette directory. Its syntax is:

```
Qd
```

Where d is the drive spec. If you omit the drive spec, the zero drive will be assumed. If you enter an erroneous drive spec, your entry will be ignored. If you enter a drive which is not in your system, the command will time out after about 10 seconds and you will receive another query (2).

If you have read in file(s) and want to begin a writing operation, respond with <ENTER> (i.e. just depress the <ENTER> key without entering any other character).

In order to read in a disk file (response to query (2) with "D"), you will be prompted for the filespec via the query:

ENTER INPUT FILE FILESPEC >

Enter the filespec to begin the read operation. This utility will default the filespec to an extension of CMD if you leave the file extension blank. If any disk I/O error results, or any disk problem that results in the file not being read to completion, you will be returned to query (2) and no fragment of the file will be added to the memory buffer. A disk file that is reread will properly append any file previously read in.

In order to read in a cassette file, you will be prompted to ready the tape with:

READY CASSETTE AND DEPRESS <H,L>

Depress the <H> (1500 baud) or <L> (500 baud) key after you have prepared the tape for input. There is no need to enter a file name. The next program found on the tape will be read. If a CHECKSUM ERROR is detected during the tape read operation, you will be prompted with the message:

TAPE CHECKSUM ERROR DETECTED - REREAD TAPE!

Any previously read in file will not be destroyed. The partial tape load will be ignored and subsequent reads will properly append any previously read in file.

If during an input, the file being loaded will exhaust your machine's memory, this message will appear:

OUT OF MEMORY!

Again, no file or files previously read into the memory buffer will be disturbed. You can proceed to save the buffer contents prior to the file that exhausted your machine's memory.

If you attempt to read in a file that is not a 'COMMAND' or 'SYSTEM' file, you will most likely receive the message:

REQUESTED FILE IS NOT A COMMAND OR SYSTEM FILE!

The read operation will cease. You will be returned to query (2).

As a disk file is read, each block detected will produce the message:

BLOCK LOADS FROM XXXX TO XXXX

At the conclusion of the file read, whether from disk or tape, the transfer address (the program address that is jumped to after loading to begin its execution) is displayed as in:

TRANSFER ADDRESS (ENTRY POINT) IS XXXX

At this point, you will recycle to the INPUT FROM DISK OR TAPE query (2).

Query (3):

PROGRAM LOADS FROM BASE ADDRESS XXXX TO XXXX

ENTER NEW BASE ADDRESS OR <ENTER> >

Query (3) will be output if one or more files were input from disk or tape. If you do not need to offset the output file) just depress the <ENTER> key and proceed to query (7). In general, if you are transferring a SYSTEM tape file to disk, and the tape file would ordinarily overlay the operating system's resident program (4200H-51FFH), you cannot load the disk file into memory from disk unless it is offset from the resident system. Once in memory, a block move routine can restore it to its original load point.

The Command File Utility will not offset any part of a load module that loads below 4200H. This is to permit programs that purposely affect system variables or display messages to the memory mapped video (3C00H-3FFFH) to load properly.

If you have input a program file that loads below 4200H and you are requesting to OFFSET the program, the following message will be displayed:

Program loads below 4200H
Enter Address to restrict offset or <ENTER> >

This gives you the option of restricting the offsetting operation below a specified address. For instance, if the program loaded a message directly to the screen, it would have a load block within the range 3C00H-3FFFH. You can maintain that load block in its original location (to the screen) by entering the lowest address above the screen area as identified in the ADDRESS LOAD LOG in response to the above query. This would provide the offset to any portion of the program originally loading at an address greater than the screen end (3FFFH) and maintain the original load addresses for any block loading into an area below the address entered.

For example, the ADDRESS LOAD LOG begins with:

Block loads from 3C00 to 3C7F
Block loads from 5200 to 5282
Block loads from 5283 to 5304

The entire program module can be offset starting at 5200 by entering "5200" in response to the "Enter Address to restrict offset or <ENTER> >" query. In this manner, the load address of the load block addressed to the screen memory will be retained as 3C00 to 3C7F.

The Command File Utility has been further improved to read the SYS6/SYS and SYS7/SYS ISAM module of LDOS. If CMDFILE interprets the module being loaded

as one conforming to the load format of LDOS's ISAM files, then the query:

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

will be displayed. If you enter the 2-character overlay number, CMDFILE will read only the desired overlay into its memory buffer. If you respond with "FF", then the entire module will be loaded. There is no attempt in the CMDFILE documentation to explain the LDOS ISAM file structure.

If you want to change the load addresses of the output file (offset it), enter the new base load address. For example) if the existing load is from 4300H to 5000H and you want it to load starting at 5300H, enter the base address 5300H. After entering the new base address, you will receive the query:

Query (4):

```
DO YOU WANT TO ADD THE OFFSET DRIVER ROUTINE (Y,N,E,C)? >
```

A response of "E" will EXIT the program, while "C" will cancel the request and return you to the main query. If you do not want the restoring driver routine appended, respond with "N" and proceed to query (7), otherwise respond "Y".

It may not be immediately obvious why you would want to offset a file but not add the appendage. One use would be to change the base address of a relocatable driver routine. Another would be to change the load address of "Tiny PASCAL" files.

Query (5):

```
DO YOU WANT TO DISABLE THE INTERRUPTS (Y,N,E)C)? >
```

A response of "E" will EXIT the program, while "C" will cancel the request and return you to the main query. If you want to disable the interrupts (which should be done if the program does any tape operation or will overlay the disk operating system's interrupt processing routine), then respond "Y", else "N". The next query is:

Query (6):

```
DO YOU WANT TO DISABLE THE KEYBOARD DEBOUNCE (Y,N,E,C)? >
```

A response of "E" will EXIT the program) while "C" will cancel the request and return you to the main query. If you want to disable the keyboard debounce routine (which should be done if the output file will overlay the disk system's debounce routine between approximately 4300H and 4400H), respond with a "Y", else respond with "N". Query (7) will now be bypassed, as the driver routine appendage dictates the transfer address. Proceed to query (8).

Query (7):

ENTER NEW TRANSFER ADDRESS OR <ENTER> TO USE XXXX >

If you want to change the transfer address (entry point), you can enter the new address. This is useful when appending two or more files since the transfer address used would default to the transfer address of the last file read in unless otherwise specified. If you had requested the driver appendage, you wouldn't be able to change the transfer address (entry point).

Query (8):

OUTPUT TO DISK OR TAPE (D,T,E,C) OR <ENTER> TO RESTART? >

Again, a response of "E" will EXIT the program, while "C" will cancel the request and return you to the main query. Just depressing <ENTER> will also return you to the main query. Cancellation is available if you do not want to create an output file but rather want just to determine disk files' load addresses.

If you want to create an output disk file, respond with a "D". You will be prompted for the filespec with:

ENTER FILESPEC TO WRITE OUTPUT >

After entering the filespec (remember CMD will be used as a default extension), the output file will be written to disk (using VERIFY).

If you want to create an output tape file, respond with a "T". You will be prompted to enter the filename with:

ENTER TAPE FILE NAME >

After entering the filename (up to six characters), you will be prompted to ready the cassette. The tape will then be written.

At the conclusion of the disk or tape writing operation, you will receive the query:

Query (9)

MODULE WRITE IS COMPLETE - WRITE ANOTHER (Y,N,E,C)? >

The "E" and "C" responses are as before. A response of "N" will also return you to query (2). If you want to generate an additional output copy, respond with "Y". If you had selected TAPE output, you would be prompted to ready the cassette and another copy would be written using the same file name as was entered, followed by query (9). If you had selected DISK output, you would be returned to query (8) so that additional output files could be written to tape or other filespecs.

TECHNICAL SPECIFICATIONS

Appendage Driver Routine

If you requested an offset to the output file's original base load address, the following routine would be appended to the end of the program provided the new base address exceeds the old base address:

```
               ORG    NEWHI+1          ;DRIVER ORIGIN
               DI (OR NOP)             ;INTERRUPTS OFF?
               LD     HL,NEWLOW        ;PT TO OFFSET START
               LD     DE,OLDLOW        ;PT TO WHERE IT GOES
               LD     BC,ENDLOD-BGNLOD+1 ;LENGTH OF MOVE
               LDIR                    ;MOVE IT IN PLACE
               JP     OLDTRA           ;GO TO ORIG ENTRY PT
```

```
Where: NEWHI  => the highest load address after offset,
       NEWLOW => the lowest load address after offset,
       OLDLOW => the original lowest load address,
                 greater than 41FFH
       ENDLOD => the original highest load address
       BGNLOD => same as OLDLOW
       OLDTRA => the original transfer address
```

If the new base address is less than the old base address (offset towards lower memory), then the driver routine appended would look like this:

```
               ORG    NEWHI+1          ;DRIVER ORIGIN
               DI (OR NOP)             ;INTERRUPTS OFF?
               LD     HL,NEWHI         ;PT TO OFFSET END
               LD     DE,OLDHI         ;PT TO WHERE IT GOES
               LD     BC,ENDLOD-BGNLOD+1  ;LENGTH OF MOVE
               LDDR                    ;MOVE IT IN PLACE
               JP     OLDTRA           ;GO TO ORIG ENTRY PT
```

```
Where: NEWHI  => the highest load address after offset
       OLDHI  => the original highest load address
```

Appending two or more files

In order to append (concatenate) two or more files into one contiguous file, keep responding to query (2) with the "D" or "T" indicator depending on where each file resides, in order to read all the desired files into the memory buffer. When the last file has been read in, respond to query (2) by depressing <ENTER> to initiate the output cycle. Note that the transfer address jumped to on initial loading of the concatenated file would be the transfer address detected from the last file input. Thus, if you want to provide control to another address, use query (7) to modify the transfer address to one of your own choosing.

If you also have to offset the concatenated file, it cannot be done during this output writing. Complete the above procedure, thereby creating the appended file. Now re-input the appended file and offset it. This second operation will provide for your transfer address as the control point after the driver routine restores the loaded module to its original load point.

Patching programs

Programs can be patched in a manner similar to LDOS's "PATCH" command. PATCH applies program corrections at the end of a load module so that the corrected bytes will overlay the incorrect bytes during the load process. Once you are made aware of the patch code, assemble it using the Editor Assembler. You may need to employ a series of ORGs and data assembly statements (DEFBs, DEFWs, etc.). The assembled object file can now be appended to the end of the original program. During the load operation of the "patched" program, the original code is first loaded but is then overlayed by your appended "patch" code.

Transferring a disk file to tape

Any load module, written using the format shown in the technical specifications, can be transferred to tape as a SYSTEM file. This feature is especially useful to assembly language programmers developing machine language programs for commercial sale. By using a disk EDTASM, such as the MISOSYS DISK*MODified EDTASM, your assembly language development can proceed using disk I/O. Even programs whose source is too large to load into the text buffer can be assembled in segments and later concatenated into one contiguous file. The file can then be transferred to a tape cassette to create a "master" for duplication. Even if you are not an assembly language programmer, this disk-to-tape feature is very useful for making SYSTEM tape backups of your disk load modules.

Transferring a SYSTEM tape to disk

If you want to employ the tape-to-disk facility, all that is needed is to perform the input from tape and not input a second file. Just use the single file read in to output it to disk. Appending "TWO" files together is not required.

C O N V (CONV/CMD)
==========

The CONV utility will allow you to move files from a Model III TRSDOS diskette onto an LDOS formatted diskette. Two drives are required. The syntax is:

```
============================================================
|  CONV :s TO :d (parm,parm,...,parm)                      |
|  CONV partspec w/wcc:s TO :d (parm,parm,...,parm)        |
|                                                          |
|  :s  is the source drive. It cannot be drive 0.          |
|  :d  is the destination drive.                           |
|                                                          |
|  partspec and wcc are as defined in the Glossary.        |
|                                                          |
|  The allowable parameters are as follows:                |
|                                                          |
|  VIS     Convert VISible files.                          |
|                                                          |
|  INV     Convert INVisible files.                        |
|                                                          |
|  SYS     Convert SYStem files.                           |
|                                                          |
|  NEW     Convert files only if they DO NOT exist on      |
|          the destination disk.                           |
|                                                          |
|  OLD     Convert files only if they already exist on     |
|          the destination disk.                           |
|                                                          |
|  QUERY   Query each file before it is converted.         |
|                                                          |
|  abbr:   All parameters may be abbreviated to their      |
|          first character.                                |
|                                                          |
============================================================
```

The CONV utility will allow you to move all or groups of files from a Model III TRSDOS disk onto your LDOS disks. It provides many different parameters to choose the files to be moved.

PARAMETERS - (VIS,INV,SYS)
---------- ---------- ----

If none of these parameters are specified, all file groups will be considered. Specifying only one parameter will automatically exclude the other two. Thus to move all files except the SYStem files, you would specify both VIS and INV.

PARAMETERS - (NEW,OLD,QUERY)

The NEW parameter is used to move files onto the destination disk only if they do not already exist. The OLD parameter will move only those files that already exist on the destination disk.

Unless you specify QUERY=NO, CONV will ask you before each file is moved onto the destination disk. If you answer the prompt Y (yes, move the file), it will function one of two ways:

If neither NEW nor OLD was specified, you will be additionally prompted with "FILE EXISTS - REPLACE IT ?" if the file already exists on the destination disk.

If either NEW or OLD was specified, you will not see the additional prompt.

You may specify a filespec/partspec to be used to determine which files to move. Wildcard characters are also acceptable. Refer to the following examples

CONV :2 :1

This example will allow you to move all files from drive 2 onto drive 1. You will be asked before each file is moved. If the file already exists on drive 1) you will be asked again before it is copied over.

CONV :1 :0 (VIS,Q=N)

This example will move all VISible files from drive 1 onto drive 0. You will not be asked before each file is moved.

CONV /BAS:2 :0 (NEW)

This example will move only those files with the extension /BAS from drive 2 to drive 0. Because the NEW parameter was specified, only those files that do not already exist on drive 0 will be moved.

CONV $$$DATA:1 :2 (OLD)

This example will move those files that have the characters "DATA" as the fourth through seventh letters in their file name. You will be asked before each file is moved, and only those files that already exist on drive 2 will be considered.

F O R M A T
==========

This is the command that allows a diskette to be FORMATted with cylinders (tracks), sectors, and a directory, so that it may be used by the system. The syntax is:

FORMAT :d (parms,parms,...)	
The following optional parameters may be used:	
NAME=""	The name that will be given to the disk.
MPW=""	The Master PassWord assigned to the disk.
xDEN	The density that will be used to FORMAT the disk, DDEN (double) or SDEN (single).
SIDES=	The number of sides to be FORMATted, either 1 or 2.
CYL=	The number of CYLinders (tracks) that are to be placed on the disk, up to 80.
STEP=	The BOOT track STEP rate that will be put on track 0 either 0,l,2,3. These values represent the step rates in milliseconds: 5'' 0=6ms, 1=12ms, 2=20ms, 3=30ms 8'' 0=3ms, 1=6 ms, 2=10ms, 3=20ms
QUERY	will prompt you for DENSITY, SIDES, STEP, and number of CYLinders.
SYSTEM	will add system information to a previously formatted disk.
ABS	If specified, will FORMAT the disk even if the disk is already FORMATted and contains data.
abbr: QUERY=Q	

The FORMAT utility is the program that will create the proper information on a diskette so the LDOS system can read and write to that diskette. A disk to be FORMATted may be blank, or it may have already been FORMATted. Note that if the FORMAT command is to be used in a JCL file, the disk to be FORMATted must be blank unless the ABS parameter is specified.

THE NUMBER OF CYLINDERS FOR AN 8'' DRIVE WILL ALWAYS BE 77, EVEN IF SPECIFIED DIFFERENTLY WITH THE (CYL=) PARAMETER.

If all parameters are not passed in the FORMAT command line, they will default to 40 cylinders, double density, single side, step=0. To be prompted for these parameters, specify the QUERY parameter on the command line. Refer to the following examples.

FORMAT :1

Since no parameters were passed, the FORMAT command will ask for them in the following manner.

DISKETTE NAME ?
Enter the desired disk name, up to 8 characters.

MASTER PASSWORD ?
Enter the desired Master Password, up to 8 characters.

The standard FORMAT configuration that will be used is:

5'' DRIVE: DDEN, CYL=40, SIDES=1, STEP=30ms
8'' DRIVE: SDEN, CYL=77, SIDES=1, STEP=10ms

If the diskette contains data, the following message will be shown:

DISK CONTAINS DATA -- NAME=nnnnnnnn DATE=mm/dd/yy
ARE YOU SURE YOU WANT TO FORMAT IT ?

You will see the name of the disk (nnnnnnnn) and the date the disk was created. The prompt will accept either a Y (yes, go ahead and FORMAT) or a N (no, don't FORMAT this disk). Answering N will return you to the LDOS READY prompt.

If the disks contain an incomplete or non-standard FORMAT, one of the following messages may appear in place of the NAME=nn message;

UNREADABLE DIRECTORY

NON-STANDARD FORMAT

NON-INITIALIZED DIRECTORY

At this point the actual FORMATting will begin. The CYLinders will be FORMATted one at a time. When all CYLinders are FORMATted, they will then be verified. If any track number appears with an asterisk (*), that track is flawed and could not be properly FORMATted.

```
FORMAT :1 (NAME="LDOS",MPW="PASSWORD",Q)
```

This command line will pass the disk Name and Master Password to the FORMAT command, and they will not be asked for in the FORMAT.

Since the Q parameter was specified in the command line, you will be prompted with the following questions:

SINGLE DENSITY?

This question must be answered Y (yes, FORMAT in single density) or N (no single density, use double density).

DOUBLE SIDED ?

This question may be answered Y (yes, FORMAT both sides) or N (no, FORMAT only one side). If your disk drives are not double sided, or if you do not have actual double sided diskettes, then this question must be answered N.

IMPORTANT: The disk drive and cable must treat the double sided drive as a single drive, not as two separate drives! A drive that treats each side of the disk as a separate drive will NOT work.

NUMBER OF CYLINDERS ?

This is the number of CYLinders (tracks) that will be FORMATted onto the diskette. Normal 5'' FORMAT is 40 tracks, although you may specify any number between 2 and 80. If a larger number is specified, the disk drive must be capable of stepping to the highest track. This question will not be asked for 8'' drives, as 77 CYLinders is assumed.

BOOT STRAP STEPPING RATE (6, 12, 20, 30/40 MSECS) ?

NOTE: The stepping rates shown are for 5'' drives. The corresponding 8'' rates are 3,6,10,20. This is the stepping rate for drive 0 that will be read in from the disk's BOOT track each time the system is powered up or BOOTed. This step rate may be changed with the SYSTEM (DRIVE=d,STEP=n) parameter, and stored in a CONFIG file created by the SYSTEM (SYSGEN) parameter.

****** IMPORTANT ******

THE BOOT STRAP STEP RATE HAS NO EFFECT ON ANY DRIVE OTHER THAN DRIVE 0.

This stepping rate will remain in effect unless it is changed by a stored configuration or with the SYSTEM command. CAUTION: too low of a stepping rate may prevent the disk from being BOOTed!.

The next question you may see is:

DISK CONTAINS DATA -- NAME=nnnnnnnn DATE=mm/dd/yy
ARE YOU SURE YOU WANT TO FORMAT IT ?

If the disk contains data, this message will appear, Answer Y (yes) FORMAT the disk) or N (no, don't FORMAT this disk).

The FORMAT command will now show each CYLinder number as it is being FORMATted. It will then verify each CYLinder, showing with an asterisk each CYLinder that it finds flawed.

The SYSTEM parameter of the FORMAT command will allow you to add SYSTEM information to a partitioned hard disk after it is formatted.

H I T A P E
==========

The HITAPE utility will permit the use of high speed (1500 baud) cassette I/O in the LBASIC and CMDFILE programs. The syntax is:

```
============================================================
|  HITAPE                                                  |
|                                                          |
|  No parameters are required                              |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

Due to space constraints and our desire to provide a high level of sophistication through the proper use of interrupt tasks, it was necessary to disable the use of 1500 baud cassette loading in the resident LDOS system. We still wanted to have the 1500 baud tape capability in the system, so a small utility was added. The utility is called HITAPE/CMD and is invoked by simply typing HITAPE <ENTER> at the LDOS Ready prompt. You may then use 500 or 1500 baud tapes in the normal manner. If HITAPE is in when a SYSTEM (SYSGEN) is performed, it will be saved with the configuration file. Both CMDFILE and LBASIC allow the use of high speed cassette only if HITAPE has been executed. If HITAPE has NOT been executed and a 1500 baud tape load is attempted, the tape will not load. It may be necessary to depress the <BREAK> key to regain control of the system.

L C O M M
=========

The LCOMM utility is a sophisticated program that provides communications capabilities between two TRS-80 systems, between a TRS-80 and a Bulletin Board System such as Forum-80, PSBBS, or other similar systems, or between a TRS-80 and a large timesharing system such as The Source (tm), MicroNET (tm), or other main-frames. LCOMM provides the capabilities of keyboard send/receive, automatic spooling to a printer through a dynamic memory buffer, and the transfer of files from system to system, without the need for handshaking when operating at 300 baud. The syntax of the LCOMM command is:

```
============================================================
|  LCOMM devspec (XLATE=X'aabb')                           |
|                                                          |
|       devspec is a valid LDOS active device,             |
|       usually the *CL, SET to the RS232T driver.         |
|                                                          |
|       XLATE specifies a one-character                    |
|       translation table option.                          |
|                                                          |
|       aa = the character to be translated.               |
|       bb = what "aa" will be translated to.              |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

CAUTION: To use any of the special functions involving the <CLEAR> key, you must have SET *KI to the KI/DVR program before entering LCOMM!

LCOMM does not "talk" directly to the RS-232 hardware, but rather to a device identified in the system's Device Control Block tables (DCB) that has been previously coupled to the RS-232 hardware through an appropriate software driver. The device LCOMM will interface with is then passed as a device specification in the command line. The device name normally utilized for this purpose is "*CL", an acronym for "Communications Line", although any other device name could be used. However, throughout this section, the *CL device will be used for reference purposes.

It is imperative that the SPOOLer not be in use while using LCOMM since the SPOOLer shares the same task slot as LCOMM (LCOMM has its own spool buffer). It is also useful, when receiving large files, to CREATE a file for receiving so that the file space is already allocated. This reduces the system overhead while running LCOMM.

LCOMM provides many user options. It interfaces with the user by utilizing the top row of keys as Programmed Function (PF) keys used in concert with the <CLEAR> key - just as the KeyStroke Multiplier (KSM) Filter uses the <CLEAR> key to provide special functions with the A-Z keys. Since most of the top row of keys are used in both their shifted and unshifted form, a brief user menu is provided to aid in jogging your mind until you become familiar with the programmed functions.

This menu can be displayed by simultaneously depressing the <CLEAR><SHIFT><?> keys.

FROM THIS POINT ON KEYS INCLUDING THE <CLEAR> AND <SHIFT> KEYS WILL BE REPRESENTED BY:

<CLR> - - WILL REPRESENT THE <CLEAR> KEY.

<CTL> - - WILL REPRESENT THE <LEFT SHIFT><DOWN ARROW> KEYS.

<SH> - - WILL REPRESENT THE <SHIFT> KEY.

< > - - WILL REPRESENT THE ACTUAL KEY THAT IS TO BE USED.
SUCH AS <!>, <#>, <%>, <6>, <9> etc.

Some of the PF keys are used to select logical devices so as to be able to turn them "on" or "off" - indicating whether the device should be acceptable for I/O. Other PF keys control the selection of parameters associated with filespecs Still others control additional functions which aid in the interface between two communicating users. The following explanations should be read and understood fully before attempting to use the LCOMM program.

The LCOMM keyboard also defines additional characters not normally available on the TRS-80 keyboard. These additional characters can be generated by pressing the indicated keys. Make note of the following keyboard enhancements:

HEX CODE	CHARACTER	GENERATED BY PRESSING	
X'1B'	<escape>	SHIFT-UP ARROW	
X'1C'	<file separator>	CONTROL-SEMICOLON	<CTL><;>
X'1D'	<group separator>	CONTROL-PERIOD	<CTL><.>
X'1E'	<record separator>	CONTROL-SEVEN	<CTL><7>
X'1F'	<unit separator>	SHIFT-CLEAR	<SH><CLR>
X'5B'	<left bracket>	CLEAR-COMMA	<CLR><,>
X'5C'	<reverse slash>	CLEAR-SEMICOLON	<CLR><;>
X'5D'	<right bracket>	CLEAR-PERIOD	<CLR><.>
X'5E'	<caret>	CLEAR-SEVEN	<CLR><7>
X'5F'	<underscore>	CLEAR-EIGHT	<CLR><8>
X'7B'	<left brace>	CLEAR-SHIFT-COMMA	<CLR><SH><,>
X'7C'	<vertical bar>	CLEAR-SHIFT-PLUS	<CLR><SH><+>
X'7D'	<right brace>	CLEAR-SHIFT-PERIOD	<CLR><SH><.>
X'7E'	<tilde>	CLEAR-SHIFT-SEVEN	<CLR><SH><7>
X'7F'	<delete>	CLEAR-SHIFT-EIGHT	<CLR><SH><8>

LCOMM will generate a modem break (extended null) if you press the BREAK key. To produce a normal TRS-80 "break" code (X'01'), press <CTL><A>. A local CLEAR SCREEN function is also available, and can be requested by pressing the <CLR></> keys.

The following PF keys are used to select appropriate devices:

*KI .... <CLR><1>

This designates the keyboard device. When LCOMM is first entered, the *KI is in an on" state. If you desire to turn it off to avoid accidentally brushing the keyboard while you are transmitting a file, you can turn off the keyboard by <CLR><1> followed by a (CLR><->, which indicates the "off" function. While the *KI is off, all PF keys are still active.

*DO ... <CLR><2>

This designates the video display device. When LCOMM is first entered, the *DO is in an "on" state. If you desire to turn it off when, for instance, the printer has been activated, a simple <CLR><2> followed by a <CLR><-> will perform the requested function. The video display will be re-activated by a <CLR><2> followed by a <CLR><:>.

*PR ... <CLR><3>

This PF key references the printer device. When LCOMM first initializes, this device is off. If you want to direct the communications transactions to your printer, do a <CLR><3> followed by a <CLR><:>. Output being routed to the printer is fully buffered through dynamic memory buffers. Therefore, it is not necessary for your printer to be capable of operating at the communications line transmission rate. Even after transactions cease, you may discover the printer still typing away.

*CL ... <CLR><4>

This PF key references the communications line device. LCOMM initializes with *CL in an ON state. You may wish to temporarily block output from being sent to the *CL so as to be able to review a file prior to transmission. Depending on your PF switch setup, if you go to a half-duplex mode (DUPLEX=ON) after turning off the *CL, you could perform a File Send (FS) which would display the file to your screen without actually sending it to the communications line. Of course, if the distant end attempted to send to you while you had the *CL off, you would not receive their transmission.

*FS ... <CLR><5>

This designates the "File Send" device. With it, you can cause a file to automatically be transmitted to the distant end. This PF key works in concert with a number of other keys. Other PF keys exist to open a

designated file, rewind a designated file, position to the end of a designated file, and close a designated file. As with the other devices discussed, the functions available to this File Send device are activated by the two-step process of first depressing <CLR><5> followed by some other PF key appropriate to the intended function. Specific details will be presented as the other PF keys are discussed.

*FR ... <CLR><6>

This is the device to be used for either receiving a file being transferred to you, or for making a file copy of the communications line dialogue. This device will also be used to download from a bulletin board system. All of the PF keys available to the FS device are also available to the FR device. These will be discussed later. This device may be turned on or off by control characters received from the communications line if the HANDSHAKE switch is on.

This concludes the complement of device references available at the keyboard. The remaining PF keys to be discussed will be used either in conjunction with one of the device keys, or by themselves to perform a stand-alone function.

ID ... <CLR><9> use with <CLR><5> (FS) and <CLR><6> (FR)

The ID function is used with either FS or FR to designate and open the desired file. If you are going to receive a file, you will perform an FR-ID by depressing <CLR><6> followed by a <CLR><9>. You will receive the prompt:

FILE NAME:

You should enter the file specification of your choice. The system will then open the file for receiving and await your next command. At this point the file is open and ready but is NOT turned ON (see ON, <CLR><:>).

If a receive file is already open, the system will ignore your ID request and issue the warning message:

FILE ALREADY OPEN

This is to guard against inadvertently issuing another FR-ID before you have closed the last file received. The same protection is available to FS. Only one FS file can be open at a time. You should close your send file after transmitting it.

RESET ... <CLR><0>

The RESET PF key performs the function of closing either a receive file or a send file. A receive file must be closed so its directory is updated. Don't forget to turn "off" a receive file before closing it. You also must close a receive file to be able to receive a subsequent file. If a device is reset, its buffer is cleared.

ON ... <CLR><:>

This PF key is used with one of the six previously mentioned device PF keys to turn "on" the device. For instance, once you have defined a receive file to the system with the FR-ID functions, you must do a FR-ON before any data will be written to it. Before a send file will start transmitting after the FS-ID, the FS-ON must be done.

OFF ... <CLR><->

This PF key performs the opposite function of the ON key. It is used in conjunction with any of the device keys to turn off the keyboard, video screen, printer, communications line, file send, or file receive. Just remember that like the ON function, the OFF function is performed in two steps. If you want to stop the receive file from further receiving, you FR-OFF by <CLR><6> followed by a <CLR><->.

The program function keys also have second functions programmed for them. These additional functions are accessible by depressing the <CLR><SH> keys along with the specified PF key. The following explains the capabilities of these second functions.

DUPLEX ... <CLR><SH><!>

This PF key is the full-duplex/half-duplex switch. In the LCOMM ON/OFF arrangement, DUPLEX-ON designates half-duplex operation. In this mode, your video display screen will display your key entries or file transmission as it is taking place. The DUPLEX-OFF mode is a full-duplex operation. Your video display will display what you send only if the distant end echoes back to you what it receives from you. Although it may be convenient to operate half-duplex (DUPLEX-ON) when communicating with another TRS-80, it may be more useful for one of the TRS-80s to play HOST and operate in half-duplex with echo to the distant end while the distant end is full-duplex (DUPLEX-OFF). This will become more evident under the discussion of file transmission.

LCOMM initializes in the full-duplex or DUPLEX-OFF state. The PF key is also one that operates in concert with the ON and OFF keys. If you want to go to half-duplex after LCOMM initializes, you must depress the <CLR><SH><!> keys followed by the <CLR><:> keys.

ECHO ... <CLR><SH><">

This will provide the function normally undertaken by a host system. If ECHO-ON is specified, everything received from the communications line will be re-transmitted. This mode is desirable if the distant end must operate full-duplex and has no "local" copy. A caution to be observed is that if both ends are set for ECHO-ON, then the first character sent will be echoed back and forth indefinitely - at least until one end turns ECHO-OFF.

ECHO-LINEFEED ... <CLR><SH><#>

The echoing of a line feed is the desired mode if the distant end is a dumb hard copy terminal that has its own local copy but expects the line feed to be sent by the host computer. With ECHO-LF-ON, anytime a carriage return is received from the communications line, a line feed character will be sent back, and a line feed will be added to any carriage return transmitted.

ACCEPT-LINEFEED ... <CLR><SH><$>

LCOMM normally ignores the first line feed received after a carriage return. If this function is turned on, all line feeds will be accepted. This may be desirable if the distant end is another TRS-80.

REWIND ... <CLR><SH><%>

The REWIND function works only with the *FR and *FS devices (FILES). It is used to rewind either file back to its beginning. For instance, say during the transmission of a file, the transmission is aborted prior to its completion. In order to re-send it, it should be rewound to its beginning so the NRN pointer is pointing to the first record. REWIND performs that function.

PEOF ... <CLR><SH><&>

This function is used to position a file to its end. A common use would be to append to an existing receive file. If you open a file for receiving by means of the FR-ID and then do a FR-PEOF, the existing receive file would NOT be overwritten with the new data, but rather the new data received will be concatenated to the old data.

8-BIT ... <CLR><SH><)>

The 8-BIT switch is used to indicate that all 8 bits of each character received from the communications line are valid. If it is not turned on, bit 7 is stripped from each character received. Do not specify this switch unless the RS-232 word length was set to 8.

HANDSHAKE ... <CLR><SH><*>

If the handshake switch is turned on, LCOMM will respond to the following four control codes received from the communications line:

```
 X'11'  DC1  - Resume transmission
 X'12'  DC2  - *FR ON
 X'13'  DC3  - Pause transmission
 X'14'  DC4  - *FR OFF
```

The DC2 and DC4 characters function identically to the *FR ON and *FR OFF programmed function keys. DC3 causes transmission through the *CL device to be halted until a DC1 is received. Reception is not affected. You can override a DC3 with the *CL ON keyboard command.

HANDSHAKE may also be turned on with the sequence <CLR><SH><*>, followed by any non-PF key (rather than the ON key). If this is the case, any time LCOMM sends the specified character it will pause transmission until a DC1 is received or a *CL ON is issued. Typically, the <ENTER> key would be specified.

EXIT ... <CLR><SH><=>

This PF key is used to return to the LDOS command level. It does not require any ON or OFF. It is a stand-alone key. Prior to exiting LCOMM, the *FR device is checked to see if an open file exists. In the event that one does, it will be closed before the exit to LDOS is made. This little feature will protect against your inadvertent exit without overtly saving an open receive file.

MENU ... <CLR><SH><?>

This PF command will display a menu to the screen. It looks like this:

```
DUPLX ECHO  ECOLF ACCLF REWND PEOF  8-BIT        HNDSH EXIT
==1== ==2== ==3== ==4== ==5== ==6== ==9==  ==0== ==:== ==-==
 *KI   *DO   *PR   *CL   *FS   *FR   ID     RES   ON    OFF
```

Notice that the display will be left to right and in the relative positions of the KEYS which are used for the functions. This is NOT intended to be a complete menu, it is like having a built in "quick reference card". The <CLR><SH><?> may be executed at anytime. The screen display will be altered to display the menu (scrolled 4 lines), but no data will be lost as LCOMM will still be buffering the incoming characters. The buffered characters will be displayed after the menu is placed on the screen.

USAGE TIPS

The beginning LCOMM user may find it useful to make up a tape containing each key's function and place the tape directly above the keys. Avery self-adhesive removable labels may be used for this purpose. Any other label will suffice, such as Press-a-Ply. The labels can even be typed to provide a neater appearance. Your keys should be labeled as follows:

KEY	UNSHIFTED	SHIFTED
1	*KI	DUPLEX
2	*DO	ECHO
3	*PR	ECHO-LF
4	*CL	ACCEPT-LF
5	*FS	REWIND
6	*FR	PEOF
9	ID	8-BIT
0	RESET	not used
:	ON	HANDSHAKE
-	OFF	EXIT

FILE DOWNLOADING

Downloading a file is easy! First identify the filespec with FR then ID. You will be prompted for the filespec. You enter the filespec - BUT - that does not cause data to be written to the file. When you want to actually start receiving, you have to turn the file on with a FR then ON. Once the file is ON, anything coming into your computer will be directed to the file in addition to any other device which is ON. Don't forget to FR-OFF and FR-RESET when you are ready to close the file.

FILE TRANSFER

To transfer an ASCII file from one system to another (LCOMM is only useful for the transmission of ASCII files since bit 7 is stripped from every byte received), you need to provide some manual handshaking. This is because LCOMM does not provide any itself. The receiving end should prepare the receiving file with an FR-ID and appropriate filespec. The sending end should perform the FS-ID and provide the appropriate filespec. After keyboarding to each other as to the readiness, the sender should provide a 10-second pause while the receiver performs the FR-ON and does not depress any other key. The sender, after the wait, sends the file with FS-ON. The sender will typically see his disk drive stop running long before the entire file is sent. This is because the file is actually spooled through memory and will usually be read in a short time period.

If both receiver and sender are in half-duplex (DUPLEX-ON), the sender will see the file appear on the video screen at breakneck speed while it goes out the communications line at 300 baud. The sender must wait until the receiver has received the last character of the file and performed a FR-OFF otherwise the key entries will also go into the file. The receiver, after performing the FR-OFF (and maybe FR-RESET), advises the sender to re-engage keyboard contact.

The following arrangement is suggested for the most satisfactory method of operation. The sender should be set DUPLEX-OFF or full-duplex. The receiver should be set DUPLEX-ON and ECHO-ON. This will provide for the receiver sending back to the sender what actually got received - while it is being received! It should be easier for the sender to note when the file's transfer has been completed. However, the sender must still wait for the receiver to acknowledge that FR-OFF has been performed.

GENERATING SPECIAL CODES

Say you need to generate special characters not generatable using the TRS-80 keyboard. You may set up a KSM file with the special characters (use the BUILD utility to generate the codes in their hexadecimal form). Then define the special keys for A-Z using FILTER *KI KSM USING KEYS (where KEYS is your keycode file).

L O G (LOG/CMD)
==========

LOG is a program that will log in the directory track and number of sides on a diskette. The syntax is:

```
============================================================
|  LOG :d                                                  |
|                                                          |
|  :d  is any currently enabled drive                      |
|                                                          |
============================================================
```

The LOG utility will provide a way to log in diskette information and update the drive's DCT. It will perform the same log in function as the DEVICE library command, except for a single drive rather than all drives. It will also provide a way to swap diskettes in drive 0.

LOG :0 will prompt you and allow you to switch the drive 0 diskette.

The "LOG :0" command must be used when switching between double and single sided diskettes in drive 0. Otherwise, it is not needed.

NOTE: Double sided disks cannot be used to boot the system!.

P A T C H
=========

The LDOS PATCH utility is used to make minor changes or repairs to existing program or data files. The syntax is:

PATCH filespec1 USING filespec2 (YANK)	
PATCH filespec USING (information in patch format)	
filespec1	Any valid filespec. The default extension will be /CMD.
filespec2	Any valid filespec for a "PATCH format" file. The default extension will be /FIX.
YANK	Will remove the PATCH specified by filespec2 from filespec1. The PATCH to remove must have been in the X'nnnn' type format.
abbr: NONE	

PATCHing may be done directly from the command line by placing the PATCH information in parentheses following the filespec to be PATCHed. The syntax for this "Command level" type of PATCH is the same as that for a PATCH file, except that multiple "lines" may be include, separated by colons. PATCH files are normally created with the LDOS BUILD command. A word processing program may be used to create PATCH files, but the file must be saved as "pure" ASCII (with SCRIPSIT use the S,A type of save).

For PATCH to work properly, a definite structure and syntax must be observed when creating the file. Here is the syntax that is required and some examples of PATCH file structure.

THE FILESPEC:

It is desirable to use some logical method of naming PATCH files. The filename to be PATCHED could be followed by a letter or a number that would be advanced as different patches become available for the same program. For example, BASIC1/FIX, BASIC2/FIX, SYS7A/FIX and SYS7B/FIX. Although not required, it is strongly suggested that all PATCH type files use the extension /FIX. This will make it easier to use these files as that is the default file extension that the PATCH utility will use and the user will not have to enter it. For example LBASIC1:0 and LBASIC1/FIX:0 both define the PATCH file LBASIC1/FIX to the PATCH utility.

PATCH LINE SYNTAX:

. (Period)

This indicates that the line is a remark, and will be disregarded by PATCH.

"string"

If a PATCH is to contain strictly ASCII characters, the PATCH code may be entered as an ASCII string enclosed within quotation marks. This type of PATCH code representation can be used in all PATCH modes.

LDOS PATCH MODES

X'nnnn'= nn nn nn nn nn nn .....

Hexadecimal load location format. This is the format that will cause a load module to be built at the end of the file being PATCHed. This ending module will then load with the program and overlay or extend the code at X'nnnn', where nnnn is the HEX load address for the PATCH line. The PATCH to be loaded is shown as nn's, which represent the HEX bytes that are desired at the specified load address. It must be noted that this PATCH mode will extend the disk file, even if all of the PATCHing is to the "inside" of the program. Because this type of PATCH will merely be added to the end of the file to be PATCHed, it may later be removed with the YANK parameter.

Drr,bb= nn nn nn nn nn nn .....

This is the DIRECT PATCH mode. The rr is the record number to be PATCHed and the bb is the BYTE in that record where the PATCH is to begin. nn are the HEXADECIMAL bytes that are to be placed in the record, replacing those that are there. This type of PATCH line does not extend the file and is applied directly to the record of the file. Because no identification of the existence of this PATCH will be placed in the. file, this type of PATCH can NOT be removed by the YANK parameter.

NOTE: Applying unauthorized PATCHes of this type to your MASTER LDOS diskette may VOID your warranty.

Lbb
X'nnnn'= nn nn nn nn nn nn ...

This format will allow patching of a LIBrary overlay in SYS6 and SYS7 of LDOS. The Lbb represents the binary coded location of the desired overlay in the SYS module. X'nnnn' represents the load address of the LIBrary module, and nn represents the bytes to be PATCHed into the LIBrary module. This type of PATCH will be added to the end of the LIBrary module, extending its length.

NOTE: This type of PATCH should not normally be created by the user. Any necessary PATCHes to LIBrary commands will be issued by LDOS Support Services. Applying unauthorized PATCHes of this type to your MASTER LDOS diskette may VOID your warranty!

(YANK)

The PATCH (YANK) parameter will allow you to remove PATCHes applied with the X'nnnn' format. The following rules will be in effect:

1) The filespec of the PATCH to YANK must be identical to the filespec used when the PATCH was applied.

2) If YANK is used without a filespec, no PATCH will be removed.

3) DO NOT PATCH A FILE MORE THAN ONCE USING THE SAME FILESPEC FOR THE PATCH FILE! It will be impossible to YANK the second PATCH from the file.

Here are some examples that will show the different PATCH formats.

```
PATCH BACKUP/CMD:0 USING SPECIAL/FIX:1
PATCH BACKUP SPECIAL
```

These commands would produce identical results. The default file extensions are /CMD for the file to be PATCHed, and /FIX for the file containing the PATCH information. The PATCH information in SPECIAL/FIX might look like this:

```
.SPECIAL PATCH FOR MY BACKUP SYSTEM ONLY!
X'6178'=23 3E 87
X'61A0'=FF 00 00
```

This is an example of a PATCH using the X'nnnn' load location format. Note the comment line in the PATCH file. This line will have no effect on the PATCH.

```
PATCH SYS2/SYS.PASSWORD USING TEST/FIX
PATCH SYS2/SYS.PASSWORD TEST
```

Note the abbreviated syntax of the second example. The USING and default /FIX extension are not necessary. The information in the PATCH file TEST/FIX might look like this:

```
.This will modify the SYS2 Module
D0B,49=EF CD 44 65
D0B,52=C3 00 00
.EOP
```

This is an example of the Direct PATCH mode. It will PATCH the specified record and byte in the file SYS2/SYS. There are 2 comment lines in this PATCH file. Neither will have any effect on the PATCH.

```
PATCH SYS6/SYS LIB1
```

This command will PATCH the SYS6 Library module. The PATCH file LIB1/FIX might contain the following information:

```
L54
X'4987'=32 20 DE AF 00 C3 66 00
```

This patch is in the Library overlay mode.

```
PATCH MONITOR/CMD:0 (X'E100'=C3 66 00 CD 03 40)
```

This command would PATCH the file MONITOR/CMD, replacing the 6 bytes starting at X'E100' with the PATCH code specified in the command line.

```
PATCH MONITOR/CMD:0 (D01,13=4C:D02,3E=66)
```

This command would PATCH the file MONITOR/CMD in two places. It uses the Direct mode to apply the PATCHes to the file's disk sector 1, relative byte 13, and disk sector 2, relative byte 3E. Note the colon (:) separating the two PATCH lines.

The PATCH utility is included so the user may modify data files and programs. THE PATCH UTILITY WILL ALSO BE USED TO MAINTAIN THE LDOS OPERATING SYSTEM. UNDER NO CIRCUMSTANCES SHOULD PATCHES BE APPLIED TO LDOS WITHOUT THE AUTHORIZATION OF LDOS SUPPORT SERVICES. TO DO SO MAY VOID YOUR WARRANTY AND MAKE IT IMPOSSIBLE FOR OUR CUSTOMER SERVICE PERSONNEL TO ASSIST YOU WITH ANY PROBLEMS YOU MAY ENCOUNTER. In other words, don't PATCH it unless YOU are willing to support it after it is PATCHed.

R E P A I R (REPAIR/CMD)

REPAIR :d (ALIEN)
:d is any currently enabled drive.
abbr: NONE

REPAIR is a program that will perform the following functions.

1) Update the DAM (Data Address Mark) for the directory track to an X'F8'.
2) Read enable DIR/SYS.
3) Check and correct the excess cylinder byte.
4) Set the grans/cylinder byte (GAT + X'CC').
5) Strip the high bit from disk track 0, sector 0, byte 3 (Directory track byte)
6) Write an LDOS boot sector onto the disk.

This program will update LDOS versions earlier than Model I 5.0.2 to correct the DAM. It will also perform all 4 functions on Model I TRSDOS 2.3 disks. The REPAIR utility MUST be done to make these disks readable by the Model III. However, be aware that a REPAIRed TRSDOS disk will not be readable on the Model I by TRSDOS 2.3. Model I LDOS owners will have information on how to make TRSDOS 2.3 read either the old or the new DAM. Compatibility with operating systems other than TRSDOS is not guaranteed.

***** IMPORTANT *****

The REPAIR utility should NOT be used on Model III TRSDOS. Use the CONV utility to remove programs and data from a Model III TRSDOS diskette.

At present, the (ALIEN) parameter is the only parameter allowed. The REPAIR program may be expanded in the future to perform other functions involving the directory.

QFB

This utility is designed to allow for a backup with format to be performed. Only floppy drives may be used, and the backup performed must be mirror image. The syntax is:

```
============================================================
|  QFB :s :d (parm,parm,parm)                              |
|                                                          |
|  :s   is the Source drive. The colon is optional.        |
|  :d   is the Destination drive. The colon is optional.   |
|                                                          |
|  The following optional parameters may be used:          |
|                                                          |
|  ALL=    parameter used to specify whether all cylinders |
|          of the source disk will be read and copied to   |
|          the destination disk, or only allocated         |
|          cylinders will be used. The switch ON or OFF    |
|          may be specified, with the default being OFF.   |
|                                                          |
|  V1=     parameter used to specify whether or not a      |
|          verify of the destination disk is to be         |
|          performed on the 1st pass. The switch ON or OFF |
|          may be used, with the default being ON.          |
|                                                          |
|  V2=     parameter used to specify whether or not a      |
|          verify of the destination disk is to be         |
|          performed on the 2nd pass. The switch ON or OFF |
|          may be used, with the default being OFF.         |
|                                                          |
|  QUERY=  Query for parameters not specified. The switch  |
|          ON or OFF may be used. The default is OFF       |
|                                                          |
|  abbr:  ON=Y, OFF=N, QUERY=Q, ALL=A                      |
|                                                          |
============================================================
```

The QFB (Quick Format and Backup) utility will allow for the creation of a mirror image backup of a source disk without having to format the destination disk prior to executing the backup. The normal means by which a mirror image backup is made using LDOS is to first format a diskette using the FORMAT utility, and then use the BACKUP utility to perform the backup. The limitations of the QFB utility are as follows:

1.) Two distinct floppy drives must be used.
2.) The source diskette must have been formatted using the LDOS 5.1.x FORMAT utility, and cannot contain any non-standard format.
3.) QFB will run exclusively on LDOS 5.1.x, versions 5.1.3 or later.

QFB will perform a "single pass" format and backup. If QFB is entered with no drives specified, prompts will appear for them. If drive numbers are specified, the first drive number will represent the source drive, and the destination drive will be the second drive number. If no parameters are specified, the defaults will be used.

Consider the results of entering the following command.

```
QFB 1 2
```

Drive 1 will be used as the source drive, while drive 2 will be the destination drive. Prior to QFB performing any action, a prompt will appear to load the diskettes. Once the proper diskettes have been installed, press <ENTER>, and the backup will begin. The following actions will take place.

1.) The source diskette will be logged in, to determine the type of format.
2.) Cylinder 0 of the destination diskette will be formatted.
3.) If cylinder 0 of the source disk contains data, it will be read into memory.
4.) If cylinder 0 of the source diskette contains data, the information stored in memory (see Step 3) will be written out to the destination diskette.
5.) Cylinder 0 of the destination diskette will be verified.
6.) Steps 2-5 will be repeated for all remaining cylinders.
7.) The following message will appear after the last cylinder has been verified:

```
Duplication complete     1 disk  created

Replace destination disks and press <ENTER> to repeat
 ..<R> to restart with new parameters
   ...or....<BREAK> to exit program.
```

8.) Press <ENTER> in response to this prompt to make another mirror image backup. Press <BREAK> to abort the QFB utility. The following prompt will appear:

```
Load SYSTEM diskette and hit <ENTER>
```

Place a system diskette in drive 0 and press <ENTER>, to return to LDOS Ready.

If it is desired to use QFB again with different parameters, press <R> in response to the prompt displayed in step 7. Doing so will cause the drives to be prompted for, and prompts will appear for all parameters.

If QFB is to be restarted, or the command QFB (Q=Y) is entered, the following prompts for the parameters will occur:

```
Duplicate unallocated tracks? (Y/N)
Verify on same pass? (Y/N)
Verify on second pass? (Y/N)
```

The first prompt relates to the ALL parameter. If it is answered with <V>, all cylinders will be read from the source diskette and written to the destination diskette, regardless of whether or not the cylinder contains information. If this prompt is answered <N>, only cylinders containing information will be read and written.

The next prompt relates to the V1 parameter. If it is answered with <V>, all cylinders on the destination diskette will be verified immediately after all writes. If answered <N>, no immediate verify will be done.

The final prompt corresponds to the V2 parameter. If it is answered with <V>, all cylinders on the destination diskette will be verified upon completion of all writing to the diskette. If answered <N>, there will be no second pass verification.

If an error occurs, an appropriate error message will be displayed, and a prompt will appear requesting the course of action that is desired. During any QFB operation, the <BREAK> key will be active, and can be used to abort the process.

IMPORTANT

QFB assumes that a mirror image backup is desired, and performs no check on the destination diskette with respect to the existence of data. Any existing information on a destination diskette will ALWAYS be destroyed. Also, QFB will NOT clear the Mod Flags of files on the source diskette.

F E D - T H E L D O S F I L E E D I T O R

FED is an all-purpose, screen oriented file editor to be used with the LDOS operating system. Its wide range of capabilities make it excellent for the advanced user, but its simplicity makes it easy to use for the novice. The editor supports both Model I and III, upper and lower case, and all drive types and sizes supported by LDOS. Some points need to made concerning FED:

This is a file editor, NOT a file copier, text editor, or word processor. It is for displaying, printing, and modifying existing files. Fed works on a file level, not a track/sector level.

FED was not designed to repair damaged disks or recover lost files, but it could be used to do so by the experienced LDOS user.

You cannot create or extend files with FED, only modify existing ones.

FED is intended to run with the LDOS operating system only.

The following is a brief description of FED's capabilities:

1) Complete editing capabilities are supported, including Hexadecimal and ASCII modifying. Direct disk patching becomes a simple matter with FED. It is even possible to write machine language code directly to disk. Small changes in files can be made instantly. With FED, there is no need to read in a large source file and reassemble it just to change one character.

2) FED allows for record advancing, backspacing and positioning. You may page through a file quickly, either forward or backward. The user need not know any diskette information (density, number of sides, number of sectors per gran, etc.). The only thing that is required to use FED is knowledge of the proper filespec.

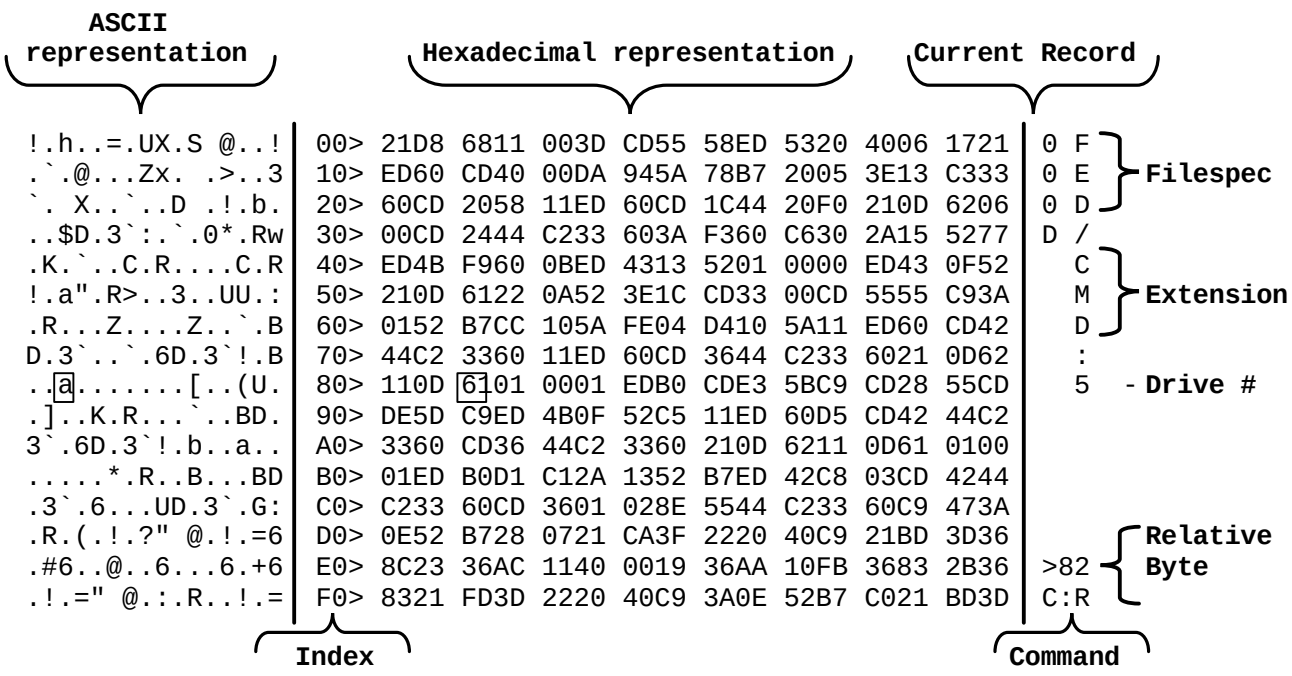
3) ASCII and Hex string searching can be performed, and a command exists which will allow you to position the cursor to the next occurrence of the search string. FED searches the entire file, not just the current edit record. It allows searching for upper/lower case ASCII strings (up to 30 characters in length)) and Hex strings (up to 15 bytes in length). FED will retain a search string, so you can go to the next occurrence of that string from the currently displayed position in the file.

4) You will be allowed to locate a Hex load address in a load module format file, and calculate the load position of a specified byte. This feature will facilitate the inspection and editing of a load module file. Just type in the load address in question, and FED will position the display to that byte. Another extremely powerful feature is the reverse of the address location command. FED will calculate where in memory a specific byte pointed to by the cursor will load. With these two features it is possible to write machine language routines directly to disk. Direct patches are made quickly and easily. Even X-patches can be installed by the experienced programmer.

5) Complete listing of a file or individual record to a printer is supported. Many safeguards have been added to make it difficult to LOCK-UP the system if a printer is deselected, out of paper, etc.

6) FED includes a 256 byte display mode, and an extended 128 byte display. Editing utilities in the past allowed for 256 byte displays only. By using this format exclusively, the variations of an ASCII/HEX display are limited. But by having a 128 character display mode, the extra space makes it more visually appealing. The filespec, drivespec, record number, input & output can be displayed horizontally instead of vertically.

Here is a sample display of the 256 byte mode:



Here is a sample display of the 128 byte mode:

```
  .K.`..C.R....C.R!.a".R>..3..UU.:.R...Z....Z..`.BD.3`..`.6D.3`!.b   - ASCII
  .a.......[..(U..]..K.R...`..BD.3`.6D.3`!.b..a.......*.R..B...BD   - Rep.

             00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F
             -------------------------------------------------
        40>  ED 4B F9 60 0B ED 43 13   52 01 00 00 ED 43 0F 52       - Hex
        50>  21 0D 61 22 0A 52 3E 1C   CD 33 00 CD 55 55 C9 3A       - Rep.
        60>  01 52 B7 CC 10 5A FE 04   D4 10 5A 11 ED 60 CD 42
        70>  44 C2 33 60 11 ED 60 CD   36 44 C2 33 60 21 0D 62
        80>  11 0D 61 01 00 01 ED B0   CD E3 5B C9 CD 28 55 CD
        90>  DE 5D C9 ED 4B 0F 52 C5   11 ED 60 D5 CD 42 44 C2
        A0>  33 60 CD 36 44 C2 33 60   21 0D 62 11 0D 61 01 00
        B0>  01 ED B0 D1 C1 2A 13 52   B7 ED 42 C8 03 CD 42 44
             -------------------------------------------------
   FED/CMD   Drive 5   Record 13  X'000D'         Relative Byte >82
         Command :R                               Values X'61'=97
```

ENTERING FED

To enter FED, simply type FED <ENTER> at the LDOS Ready prompt. Doing so will cause FED to be loaded and executed. The first prompt you will see will ask you to enter a filespec. Answer this prompt by giving the filespec of the file you wish to examine/modify. If you wish to exit FED at this point, press the <BREAK> key, and you will be returned to the LDOS Ready prompt. If an illegal or improper filespec is given, the appropriate error message will appear, and you will be allowed to re-enter the filespec. The filespec prompt may be bypassed by entering FED using the syntax: **FED filespec<ENTER>**.

After a valid filespec has been given, the FED 256 character mode will appear on the screen, and the first record (record 0) will be contained in the "edit buffer" (The term "edit buffer" will refer to the record of the file currently in the computer's memory. The edit buffer will contain one 256 byte record 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), and 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 (0-255) relative to the sector in question. Also, hexadecimal notation (X'nn') will be used to represent the current record number and relative byte number.

There will also be an input cursor located on the bottom right portion of the screen, following the message "Command". This will be referred to as the "command buffer", and will be the place on the screen where commands are entered. The current command in use will always be displayed there. When in the 128 character mode, the command buffer will appear on the lower left portion of the screen.

Also shown on the screen will be additional information which may be of importance to the user (such as current record number, filespec, relative byte within the sector, etc.). The sample displays on the previous page will show where on the screen this information will be displayed. For certain commands, inputs of several characters will be required. Depending on the mode you are in (256 or 128 character mode), these inputs will be taken in a different manner.

When in the 256 character mode, these types of inputs will be taken in an input box, and the input box will be positioned vertically along the right hand edge of the display.

When in the 128 character mode, these types of inputs will be taken directly to the right of the command buffer. No input box will appear, but a flashing cursor will be present, indicating that an input is requested.

It is advised that when using FED, the <BREAK> key should always remain enabled, as some FED commands are exited by the use of the <BREAK> key.

The remainder of this manual will be dedicated to the discussion and explanation of all commands available in the FED program.

FED LIBRARY

```
<A>          Enter ASCII character modify mode
<B>          Position to the Beginning record
<C> cccccc   ASCII Character string search for cccccc
<D>          Dump Disk File to printer (from current position)
<E>          Position to the Ending record
<F> nnnnnn   Find Hex string nnnnnn
<G>          Go to the next occurrence of last search (Hex or ASCII)
<H>          Enter Hex modify mode
<L> nnnn     Locate Hex load address nnnn
<M>          Memory location of a specified byte
<N><ENTER>   New File request (open a different file)
<O>          Output a top-of-form to printer (X'0C')
<P>          Print current record in edit buffer
<R> nnnn     Position to Record nnnn
<S><ENTER>   Save current record (sector) in edit buffer
<T>          Toggle between 256 and 128 display mode
<X><ENTER>   eXit FED and return to LDOS Ready
<Z>          "Zip" through File Load Blocks
<BREAK>      Cancel current FED command
<ENTER>      Display FED instruction set (Menu)
<;> (+)      Advance one record in the file
<->          Backup one record in the file
<SHIFT><=>   Display binary representation of byte (128 byte mode only)
```

CURSOR MOVEMENT

```
<?>          Move cursor left.
<?>          Move cursor right.
<?>          Move cursor up.
<?>          Move cursor down.
<SHIFT><?>   Position cursor to relative byte X'00' of the current record.
```

MENU DISPLAY OF FED INSTRUCTION SET

```
<;>   Forward ONE Record          <BREAK>     Cancels command
<->   Backward ONE Record         <N><ENTER>  New File
<B>   Beginning Record of File    <S><ENTER>  Save Record
<E>   Ending Record of File       <X><ENTER>  Exit FED
<R>   Position to Record          <H>   Hexadecimal Modify
<Z>   Go to next Load Block       <A>   ASCII Modify
<M>   Calculate Load Address      <T>   Toggle Display modes

<C>   Find ASCII String           <F>   Find Hex string
<L>   Locate Hex Load Address     <G>   Go next occurrence

<D>   Dump File to Printer        <O>   Output top-of-form
<P>   Send Buffer to Printer      <=>   Display Binary Value
```

Press <ENTER> to Return to Display Mode

FED MANIPULATION COMMANDS

<;> Advance one record sequentially in the file. For example, if FED was currently displaying record X'000C' and <;> was pressed, the contents of record X'000D' would be displayed (provided that a record X'000D' existed in the file). An "*" will be displayed directly below the record number when pointing to the last record in the file. Issuing the <;> command will not change the position of the relative byte cursors. A "+" will be shown in the command buffer to show positive motion in the file.

<-> Back up one record in the file. If FED was currently displaying record X'0087' and <-> was pressed, the contents of 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 0 is being displayed. A "-" will be shown in the command buffer to show negative motion in the file.

<B> Position to the beginning of the file (record X'0000') and point cursors to relative byte X'00'.

<E> Position to the Ending record of the file. An "*" will appear directly below the record number, indicating that the record being displayed is the last record in the file. The relative byte cursors will be positioned on the last byte in the file (not necessarily relative byte X'FF'). Since LDOS uses sector I/O, the whole sector will be displayed, and any byte in the sector may be modified. Realize that any modifications made to bytes beyond the last byte will not cause the EOF marker of the file to be updated to reflect these changes.

<R>nnnn Position to record X'nnnn', provided record X'nnnn' exists in the file. If the record does not exist, an "*" will appear in the command buffer. After entering <R>, a box will appear below the record number display box. The input for the record number to retrieve will be taken in this box. Hex digits (0-F) must be entered, as any other characters will be ignored. You may press <BREAK> to 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 <R> <2> <1> <ENTER>. To position to record X'0007', type <R> <7> <ENTER>. The position of the relative byte cursors will remain unchanged after the new record is retrieved.

<Z> Points the cursors to the next "Type" byte (X'01', X'02', X'05', X'07', X'10', X'1F') of a Load Module File. This feature is designed to allow the user to ZIP through machine language files quickly. Place the cursors on a "Type" byte and press <Z>. After this has been done, the cursors will be positioned over the next "Type" byte. Encountering a X'02' will terminate a <Z>ip. Any string searching, address locating, or address calculating will disable an active <Z>ip. For more information on "Type" bytes, refer to FILE FORMATS in the Technical Information section of the LDOS manual.

FED MODIFICATION COMMANDS

<A> Enters the ASCII Modify Mode. In this mode, modifications can be made in ASCII. Anything you can type in from the keyboard (with the exceptions of the <BREAK> key and the arrow keys) can be sent to the edit buffer. Modifications can be made by positioning the cursor over the bytes to be changed. After the A command is issued, the command buffer will display an "A". From this point on, any characters entered will be taken as modifications to the bytes in the record. The arrow keys may be used to position the cursor for additional edits. To exit the ASCII modify mode, the <BREAK> key must be pressed.

To modify a byte: 1) Position the cursor to the desired byte to change. 2) Type in the ASCII character to replace the original. After making a modification, the relative byte cursors will move to the next byte of the record. Note - no changes are made to disk, only to the edit buffer. To make changes to disk, see the <S>ave command.

<H> Enter the Hex Modify Mode. In this mode, the user can modify bytes in the currently displayed record. Modifications can be made by positioning the cursor over the bytes to be changed. After the H command is issued, the command buffer will display an "H". From this point on, any characters entered will be taken as modifications to the bytes in the record. The arrow keys may be used to position the cursor for additional edits. To exit the Hex modify mode, the <BREAK> key must be pressed.

To modify a byte: 1) Position the cursor over the desired relative byte in the record. 2) Enter the hex digits that you wish to overwrite the current information with. As digits are entered, the previous hex digits will be replaced by the digits entered from the keyboard. The first hex digit entered will modify the first hex digit in the byte, and the second hex digit entered will modify the second hex digit in the byte. After an entire byte has been modified, the cursors will move to the next byte in the record. Note - no changes are made to the disk, only to the edit buffer. To make changes to disk, see <S>ave.

<S><ENTER> Save the contents of the current edit buffer to disk. The current record pointed to by FED will be overwritten by the contents of the edit buffer. Any changes made after the initial read of the record will be written to disk.

FED SEARCH COMMANDS

NOTE: The search commands described below may cause the information in the edit buffer to be overwritten by information contained in subsequent records of the file. If edits have been made to the information in the edit buffer, they should be saved to the disk prior to issuing a search command. In most cases, you should issue a "B" command prior to performing a search. This will assure that the entire file will be searched, and no occurrences of the search string will be missed.

<C>cccccc Find ASCII string "cccccc". Issuing the <C> command will cause a search to be performed for the string (cccccc). The search will start at the relative byte pointed to by the cursors. The search is identical to the <F>ind Hex string command, except that the search criteria is an ASCII string of 1 to 30 characters (depending on the display mode being used). Also, the number of characters to be searched for may be an even or an odd number. See the <F> command for further information.

<F>nnnnnn Find hex string "nn nn nn". The <F> command will perform a search for the hex string nn nn nn, starting at the relative byte pointed to by the cursors. (If in the 256 byte display mode, the length of the hex string may be from 2 to 6 characters long, and must be represented as an even number of characters. If in the 128 byte display mode, the length of the hex string may be from 2 to 30 characters long, and must be represented as an even number of characters). The search will begin from the byte over which the cursor is positioned, and will scan all records past the current record until the first occurrence of the string is encountered. If a match is found, the record containing the match will be displayed, and the cursors will be positioned over the first character of the record which matches the search string. To terminate any search, you may press the <BREAK> key. This will cause the record which was contained in the edit buffer prior to the search to be read back in from the disk. If a match is not found, an "*" will appear in the command buffer, and the cursor will be positioned over relative byte X'FF' of the last record. Only hex bytes can be entered, not hex digits. An "*" will appear in the command buffer if an odd number of hex digits are entered. If there are multiple occurrences of the specified string, you can "go" to each occurrence by means of the <G>o command.

<G> Go to the next occurrence of current search criteria (string or "L" address). The <G>o command performs a continuation of the last search. If the last search was for a string, it will go to the next occurrence of that string. If the last search was for an address, it will <G>o to the next occurrence of that address. Note - the <G>o works in conjunction with the last search! If the data searched for is not found, one of two things will happen. If the <G>o command is issued after an <L> command and the address is not located, the current record will be read in from disk, and the position of the relative cursors will be unaffected. If the <G>o command is issued after any other search command and the search criteria is not located, the last record will be displayed with the cursor pointing at relative byte X'FF'.

<L>nnnn Locate Hex load address X'nnnn'. The <L> command allows the user to find load address X'nnnn' in a load module file. Unlike the string searches, the <L>ocate command starts its search at record X'0000', rather than at the current cursor position. If the address is located, the record containing the byte at that load address will be displayed, and the cursors will be positioned over this byte. If the address is not located, an error message will be displayed, and you will be prompted to press <ENTER> to continue. After <ENTER> is pressed, the record which was in the edit buffer prior to issuing the <L> command will be retrieved, and the position of the cursors will be unaffected. If a <L>ocate is performed on a non-load module file, the appropriate error message will be displayed. The <G>o command may also be used in conjunction with the <L> command to locate multiple occurrences of the same load address.

FED OUTPUT COMMANDS

<D> List the file to the printer, in the same format as the <P>rint command. The <D> command will print all records in the file, starting from the current record number. All records to be printed will be read in from the disk. To halt the printing prior to its completion, depress the <BREAK> key. After the printing has been completed (or terminated), the record which was in the edit buffer prior to printing will be retrieved from disk and stored in the edit buffer, and the cursor position will remain unaffected. Realize that if changes have been made to the record in the edit buffer) these changes should be saved to the disk prior to issuing the <D> command. Several precautions have been taken to prevent computer lock-up during the printing of records. If the printer should become disabled for some reason during printing, FED will continue the printing process after the printer has been enabled. Please note that the LDOS spooler will work in conjunction with the printing operations of FED. Also note that all records will be printed in 20 lines, with a spacing of 2 lines between records. This will allow 3 records to be printed on 66 line/page paper.

<0> Output a top-of-form character (X'0C') to the printer.

<P> Send edit buffer contents to a printer in ASCII and Hex. The <P> command will print the contents of the edit buffer. After the <P> command has been issued, the record display on the screen will be sent to the printer. To terminate printing at any time, depress the <BREAK> key. The following is a sample of the output produced by the <P> command:

```
                SPACE/CMD   DRIVE 1   RECORD 22  X'0016'

0123456789ABCDEF  BYTE  00 01 02 03 04 05 06 07   08 09 0A 0B 0C 0D 0E 0F
================================================================================
<.2<.:..<2....D.  <00>  3C 09 32 3C 7F 3A 04 7F   3C 32 04 7F C3 F6 44 A5
GAME OVER PLAYER  <10>  47 41 4D 45 20 4F 56 45   52 20 50 4C 41 59 45 52
 < >NEW HIGH SCO  <20>  20 3C 20 3E 4E 45 57 20   48 49 47 48 20 53 43 4F
REEN....TER NAME  <30>  52 45 45 4E 01 00 B4 97   54 45 52 20 4E 41 4D 45
              !.  <40>  20 20 20 20 20 20 20 20   20 20 20 20 20 20 21 C4
W:`...Ww:a...Ww:  <50>  57 3A 60 7F CD F1 57 77   3A 61 7F CD F7 57 77 3A
a...Ww:b...Ww:b.  <60>  61 7F CD F1 57 77 3A 62   7F CD F7 57 77 3A 62 7F
..Ww....0#.././.  <70>  CD F1 57 77 C9 E6 0F C6   30 23 C9 CB 2F CB 2F CB
/./...0#.  PLAYE  <80>  2F CB 2F 18 F0 C6 30 23   C9 20 20 50 4C 41 59 45
R < >  .........  <90>  52 20 3C 20 3E 20 20 FF   FF FF FF FF FF FF FF FF
................  <A0>  FF FF FF FF FF FF FF FF   FF FF FF FF FF FF FF FF
................  <B0>  FF FF FF FF 80 88 B7 B7   B7 B7 9D 80 AE BB BB BB
.......... INTRU  <C0>  BB 84 80 80 80 80 80 80   80 80 20 49 4E 54 52 55
DERS............  <D0>  44 45 52 53 AE 9D AE 9D   88 9B A7 84 88 9E AD 84
....0xH.x.0|H0.H  <E0>  A0 99 A6 90 30 78 48 B4   78 84 30 7C 48 30 F8 48
POINTS20 POINTS1  <F0>  50 4F 49 4E 54 53 32 30   20 50 4F 49 4E 54 53 31
```

FED MISCELLANEOUS COMMANDS

<ENTER> Display FED instruction menu.

<X><ENTER> Exit FED and return to LDOS Ready.

<N><ENTER> Open a New file for editing. A prompt for the filespec will be displayed. If you input an invalid or improper filespec, an error message will appear, and you will be allowed to re-enter the filespec. Note that FED will never close files, as files need not be closed with this type of editor.

<BREAK> Clear command buffer. Pressing <BREAK> will cancel any partial command, and will cause the termination of any command being executed. It is also the only way to exit the ASCII and Hex modify modes. Anytime there is any doubt as to the operation being performed by FED, you may press <BREAK>, and the command buffer will be cleared.

<SHIFT><=> Display binary representation of byte pointed to by the cursors. This command may only be used when in the 128 character mode, and will be ignored if issued in the 256 character mode. After depressing <SHIFT><=>, 8 binary digits will be displayed next to the command buffer. For example, if the cursors were positioned over relative byte X'27', and this byte of the edit buffer contained a X'F3', the binary digits 1111,0011 would be displayed.

<M> Calculates the address in memory where the byte pointed to by the cursors will load. This command works with load module format files only. If the byte is contained in a load block, the load address will be displayed below the record number. If the byte is not in a load block (e.g. a comment line, file header, etc.) the error message "Byte not in load block" will be displayed.

<T> Toggle between the regular 256 byte mode and the extended 128 byte mode. By pressing <T>, the user shifts to "the other" mode. The 128 character mode has all of the same commands as the 256 character mode. The display is a window of the 256 byte record, and 128 bytes will be displayed. By moving the cursors (usually with the <UP> and <DOWN> arrows) you will notice a scrolling effect. The ASCII display will be at the top of the screen instead of the 16 leftmost columns. The current record number is displayed in decimal as well as hexadecimal. All inputs will be taken horizontally instead of vertically. ASCII and hex search inputs will allow 30 characters instead of 6.

USING DRIVERS, FILTERS, OR PROGRAMS WITH FED

FED works harmoniously with other programs as long as FED is not tampered with. User programs should not use any memory below X'7700'. When returning to FED from some other function, the display may appear to be garbaged. Simply press <BREAK> and the FED display will be re-established. As far as drivers and filters are concerned, FED uses keyboard, video and printer DCB'S, so any vectors changed by another driver will be picked up by FED. NOTE: For FED to function properly, the user must maintain standard ASCII values and restore any registers, DCB's, devices, etc. to their original values. You may also use FED in conjunction with a machine language program. It is possible (although not recommended) to use FED from within LBASIC via the CMD"FED" command. In order to utilize FED in this manner, the user should make sure that at least 10,000 bytes are free in LEASIC. If the number of free bytes is less than 10,000, the system will most likely crash.

J O B L O G (JL/DVR)
==========

This driver program will establish the LDOS Joblog device. The syntax is:

```
============================================================
|  SET *JL TO JL/DVR USING filespec/devspec                 |
|                                                           |
|  filespec/devspec is the file or device to be sent the    |
|  Joblog information.                                      |
|                                                           |
|  abbr: NONE                                               |
|                                                           |
============================================================
```

The JL/DVR program will establish the LDOS Joblog device (*JL). Once set, a log of all commands entered or received will be sent to the specified file or device, along with a time stamp. Note that the time stamp will be determined from the setting of the system's Real Time Clock (see the TIME library command). If a filespec is used, the default extension will be /JBL.

NOTE: It is not advisable to SYSGEN the *JL device if it is routed to a disk file.

SETting *JL will use high memory. The RESET *JL command will terminate the JobLog, and close any associated disk file. However, if *JL is SET again, a new high memory allocation will be made.

To view the contents of a JobLog disk file, you must first RESET *JL, so the file will be closed. You may wish to add a trailing exclamation point "!" to the end of the filespec, so that constant EOF maintenance will be invoked (see the filespec definition in the GLOSSARY). The LIST library command will allow you to list the contents of the file to the screen or to the printer.

Note that if an existing filespec is used when SETting *JL, any information sent to the JobLog file will be appended to the end of the file.

You may wish to send the information to a device such as *PR, rather than a file. In this case, a devspec rather than a filespec would be used in the command line when SETting *JL to its driver.

K I / D V R
==========

The KI/DVR program will set the LDOS keyboard driver. The syntax is:

```
============================================================
|  SET *KI TO KI/DVR (parm,parm,...)                       |
|                                                          |
|  The allowable parameters are as follows:                |
|                                                          |
|  TYPE   activates the type ahead feature.                |
|                                                          |
|  JKL    activates the screen print option.               |
|                                                          |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

The KI/DVR program allows the user to establish the type ahead and screen print options, and also activates the <CLEAR> key. This driver must be set if SYSTEM (SVC), KSM, MiniDOS, LCOMM, or any other program that utilizes the <CLEAR> key as a control key is to be used.

As this driver is established with the SET Library command, it must be applied before any other *KI filters. When first initialized, the driver will reside in high memory. If *KI is RESET and then SET again to KI/DVR, the same high memory space will be used.

However, if any parameter was not specified initially, additional high memory will be assigned for that parameter if *KI is RESET and then SET again with additional parameters specified.

The keyboard repeat and debounce features are part of the ROM keyboard driver, and will be available even if this driver is not used. However, using the KI/DVR program will provide an increased key repeat rate.

Specifying the TYPE parameter enables the Type Ahead feature. This will provide a 128 character buffer, and will allow typing ahead even when the system is performing other functions such as disk I/O. If you make a mistake while typing ahead, pressing the <CLEAR><@> keys will empty the type ahead buffer. To temporarily disable the type ahead function, use the command SYSTEM (TYPE=OFF). It may be re-enabled with a SYSTEM (TYPE=ON) command.

The screen print option will send the contents of the video screen to *PR (usually a line printer) whenever the <LEFT SHIFT><DOWN ARROW><*> keys are pressed.

M E M D I S K / D C T - DISK DRIVE IN MEMORY
==

The MEMDISK/DCT program will allow you to set up an area of memory to simulate a disk drive. This area of memory can then be accessed with any standard disk I/O commands. The SYSTEM Library command is used to install the in memory disk drive as follows:

```
============================================================
|  SYSTEM (DRIVE=n,DRIVER)                                 |
|                                                          |
|  n = the logical drive number to be used                 |
|                                                          |
|  abbr: none                                              |
|                                                          |
============================================================
```

You will then be prompted to enter the DCT driver program. Respond by typing in the name MEMDISK. Before explaining how to enable or disable MEMDISK, a brief description of the simulated disk drive is needed.

MEMDISK organization

The simulated disk drive will consist of a short disk driver program, and the actual memory allocated for the tracks. Each MEMDISK track will consist of 6 granules. The track size is adjustable, with 1 or 2 sectors per granule. Thus a track will take 1.5K or 3K, depending on the number of sectors per granule you select. Track #1 will always contain the directory, regardless of the total number of tracks on the disk. There is no space allocated for a track 0. As a result, that track will always show up as "locked out" in the free space map. The driver program will automatically take care of any system requests to locate the directory track number normally stored in sector 0 of track 0. The user is allowed up to nineteen 1 sector/gran tracks, and up to nine 2 sector/gran tracks.

Using 2 sector/gran tracks will provide 80 directory slots, with 16 of these being reserved for System (/SYS) files. When using 1 sector/gran tracks, the directory space is limited to 32 files, with only 8 of these spots reserved for system (/SYS) files. When using this smaller track size, there are system slots available for only the following /SYS files:

BOOT and DIR SYS0 and SYS1 SYS6 thru SYS9

You should not attempt to put any of the other system files onto the memory disk if you are using the 1 sector/gran tracks. They may be put into memory with the SYSTEM (SYSRES) feature (LDOS 5.1 only). Also, the file BOOT/SYS is never physically present in the drive.

Enabling MEMDISK

After typing in MEMDISK in response to the driver prompt) you will see the following display:

```
1 or 2 Sectors/Granule (0=disable) ?
```

The initial prompt will ask you to select 1 or 2 sectors per granule Using 1 sector grans will provide the most efficient storage, but will also limit the directory as explained above. After selecting the gran size, you will see the following prompt:

```
Note:  Each track equals x.xK of space.
Number of free tracks 1-nn
```

The space per track will be 1.5K for 1 sector grans, and 3.0K for two sector grans. The maximum number of tracks will be nineteen for a 1 sector/gran disk, and nine for a 2 sector/gran disk. MEMDISK will not allow an allocation which would result in HIGH$ being below x'8000'. If you enter too many tracks, you will see message "Insufficient Memory". The process will abort, and you will return to the LDOS Ready prompt with no in memory disk drive installed. You can then restart the process, and specify fewer tracks to meet the memory restrictions. If there is enough memory available, the disk drive memory will be "Formatted" and tested. You will see the following display:

```
Verifying RAM Track nn

Verifying Complete, RAM Good
Directory has been placed on Cylinder 1

Note:  Real-Time clock still accurate.
```

The initialization routine will run a brief memory check on the space allocated to the MEMDISK drive. If the verifying detects a bad memory location, the operation will abort, and the following message will be displayed:

```
Verify Error at location X'nnnn'
```

At this point, you should determine the cause of the memory error before attempting to establish the MEMDISK drive again.

NOTE: MEMDISK may be saved with the SYSTEM (SYSGEN) command.

Disabling MEMDISK

Once MEMDISK has been established, it may be temporarily disabled with the SYSTEM (DRIVE=n,DISABLE) command. The memory area will remain untouched. It may later be re-enable with a SYSTEM (DRIVE=n,ENABLE) command. The MEMDISK driver and space allocation may be removed entirely by using the SYSTEM (DRIVE=n,DRIVER) command again. At the initial number of tracks prompt) answer by entering a zero (0). However, this will only work if no other memory below the MEMDISK driver has been protected HIGH$ will be moved back to where it was before MEMDISK was established, and the DCT location for the drive will be changed to show a disabled drive. If HIGH$ is not located directly below MEMDISK, you will see the following message:

```
Unable to disable MEMDISK, additional high memory used
```

You should be sure that MEMDISK is the last program executed that uses high memory if you wish to remove it and release the memory at a later time.

R S - 2 3 2 T
==========

This Driver program will accept and configure the RS-232 hardware, using the SET LIBrary command as follows:

SET devspec TO RS232T/DVR (parm,parm,...)

devspec is the device to be used with the RS-232, normally *CL, or the Comm Line.

parm parameters used to configure the RS-232 port, and establish line conditions:

BAUD= sets the BAUD rate to any supportable rate.

WORD= sets the word length, 5 to 8 bits.

STOP= sets the stop bits, either 1 or 2.

PARITY= sets the PARITY switch, ON or OFF. If ON is specified, EVEN or ODD may also be used.

BREAK determines whether RS232T can set the system BREAK, PAUSE, or ENTER bits

RS-232 LINE CONDITIONS

Output Parameters		Input Parameters	
DTR=	ON/OFF	**DSR=**	ON/OFF
RTS=	ON/OFF	**CD=**	ON/OFF
		CTS=	ON/OFF
		RI=	ON/OFF

abbr: ON=Y, OFF=N, BAUD=B, WORD=W, STOP=S, PARITY=P

This program is a driver for the optional RS-232 board in the Model III. It allows you to set the parameters to values that match any other RS-232 devices. The receiving side of the driver is interrupt driven and contains an internal 128 character buffer to prevent loss of characters during disk I/O and other lengthy operations.

The defaults for the configuration parameters are as follows:

```
BAUD = 300
WORD = 7
STOP = 1
PARITY = ON,EVEN
```

The Line Condition parameters have been provided so that you may set up the conventions required by most communicating devices. As specified by standard RS232 conventions, a TRUE condition means a logic 0, or positive voltage. A FALSE condition means a logic 1, or negative voltage. DTR and RTS may be set to a constant TRUE by specifying the ON switch. If DSR, CD, CTS, or RI are specified ON, the driver will observe the lead and wait for a TRUE condition before sending each character. If specified OFF, the driver will wait for a FALSE condition before sending a character. If not specified, the lead will be ignored.

The BREAK parameter is provided to allow LDOS to recognize BREAK, PAUSE (Shift-@), and ENTER characters received from the communications line. This would be useful in "host" type applications. The BREAK parameter will cause RS232T to set the system break bit whenever a modem break (extended null) or an ASCII X'01' is received. The system pause bit will be set whenever the ASCII code X'60' is received, and the system enter bit will be set whenever a carriage return (X'0D') is received. If the parameter is not specified, RS232T will never set the break, pause, or enter bits.

The following examples show how these driver programs might be used.

```
SET *CL TO RS232T/DVR (BAUD=300,WORD=8,STOP=1,CTS)
SET *CL RS232T (BAUD=300,WORD=8,STOP=1,CTS)
```

This example will configure the RS-232 using the values specified. Notice that PARITY was not specified, and will default to ON, EVEN. The use of TO in the command line is optional. Also, the default file extension for the SET command is DVR. CTS was specified, so the driver will look at the CTS line for a TRUE condition before sending a character. This would be useful, for instance, if a serial printer had its BUSY line hooked to the CTS line on the computer.

The device *CL will be the device the system will use to communicate with the RS-232 hardware.

```
SET *CL TO RS232T/DVR (BREAK)
SET *CL RS232T (BREAK)
```

This example will configure the RS-232 hardware to the default values. Because the BREAK parameter was specified, certain system functions will recognize break, pause, or enter characters from the RS-232 as if they came from the keyboard.

```
SET *CL RS232T (RTS,CTS,BREAK)
```

This example is identical to the previous, except that the RTS line will be held in a constant TRUE state, and the driver will not transmit any characters unless the external device raises the CTS line.

K E Y S T R O K E M U L T I P L Y (KSM/FLT)

The program KSM/FLT allows the use of files containing pre-defined phrases associated with the unshifted alphabetic keyboard keys to be used as direct keyboard inputs. The syntax is:

FILTER *KI TO KSM/FLT USING filespec
filespec is an existing KSM type file
abbr: NONE

This device filter is linked to the keyboard (*KI) with the FILTER command. Please refer to the LIBrary command section for a complete explanation of the FILTER command.

***** IMPORTANT *****

Because the MiniDOS filter uses shifted alphabetic keys, the use of any shifted alphabetic key will NOT be allowable when using the KSM filter.

The KSM/FLT program will load up to 26 phrases from the specified file (filespec) into memory. These phrases will be taken as though they were typed in from the keyboard when the <CLEAR> key and the specified unshifted alphabetic key are held down together. The default file extension for the filespec is /KSM.

To create a KSM file, use the BUILD command in the following manner:

BUILD filename/KSM The extension of the filespec MUST be /KSM!

This BUILD command will display the alphabetic keys one at a time and allow you to input your desired phrase or command. The actual display will be:

A=>
B=>
C=>

and will continue up to Z=>. Once all 26 characters have been assigned, the file will be closed and the BUILD will be terminated. The BUILD may also be terminated any time before reaching Z=> by pressing the <BREAK> key in response to any character prompt.

The following rules will govern the entry of phrases during the BUILD.

Each phrase should be terminated by pressing <ENTER>. This does not place an <ENTER> character at the end of the phrase, but merely signifies the end of the phrase.

To embed an <ENTER> into a phrase, use the semi-colon character (;). The semi-colon will be translated into an <ENTER> whenever it is encountered in a phrase.

Length of phrases should be limited to 63 characters for LDOS command lines and 255 characters for LBASIC lines.

The BUILD (HEX) parameter may used to create characters or strings that are not directly available from the keyboard. These strings will be shown as the corresponding graphics characters when displayed.

Following are some examples of the KSM function in the LDOS command mode.

```
A=> DIR :0
```

This string would appear when the <CLEAR> and <A> keys were pressed together. The command DIR :0 would be shown but would not be acted upon until the <ENTER> key was pressed.

```
A=> DIR :0;
```

This is the same as the last example, except the DIR :0 command would be executed immediately as an <ENTER> was the last character of the phrase (represented by the semi-colon).

```
F=> FREE;DEVICE;
```

This phrase would be read in when the <CLEAR> and <F> keys were pressed together. The LDOS command FREE would be executed, and the command DEVICE would execute immediately afterwards.

Following are some examples of the KSM function in LBASIC.

```
F=> FOR
N=> NEXT
C=> CLEAR5000:DEFINT A-Z:DEFSTR S,U,V:DEFDBL D:DIM S(100);
```

The keys <F> and <N> could be assigned the phrases FOR and NEXT. Whenever these LBASIC commands were needed, they could be entered in by pressing the <CLEAR> and alphabetic key. The <C> key would insert the entire line associated with it into the program. It is possible to assign the most common LBASIC keywords and commands to a KSM file so they may be instantly inserted while programming in LBASIC.

It is not absolutely necessary to use the BUILD command with the /KSM file extension to create the KSM files. Any file in ASCII format can be used by the KSM/FLT program. If you wish to use the BUILD command without the /KSM extension, a BASIC program, or a word processor to create the KSM file, observe the following format.

When the file is read in by the KSM/FLT program, it is stored in memory according to lines. This means that all characters up to the first carriage return (X'0D') will be assigned to the letter <A>, all characters up to the next carriage return to the letter <B>, etc. If a key is to be skipped or left undefined, a carriage return must be inserted for that character. Remember that the semi-colon character will be translated into an <ENTER> by the KSM/FLT program.

Once *KI is filtered with a KSM file, a different KSM may be utilized in the following manner:

Use a RESET *KI command to remove the current KSM filter This will also remove any other ROUTE, SET, FILTER, or LINK that has been done.

If the length of the new KSM file is less than or equal to the original KSM file, re-establish the KI/DVR program and any other desired features. Now apply the new KSM file with the FILTER command.

If the new KSM file is larger than the original, a global RESET must be done first. If it is not, the message "REQUEST EXCEEDS AVAILABLE MEMORY" will appear, and the FILTER operation will abort.

M i n i D O S (MINIDOS/FLT)
==========================

The MiniDOS filter program provides a means to perform certain functions using single keystrokes. The syntax is:

```
============================================================
|  FILTER *KI USING MINIDOS/FLT                            |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

The MiniDOS filter allows the keyboard driver to intercept certain keyboard inputs and immediately act on them. *KI MUST have been previously set to the KI/DVR program if MiniDOS is to be used.

To allow the MiniDOS filter to properly intercept all keys, it must be the last filter applied to the keyboard (*KI).

Note that the MiniDOS filter will reside in high memory. If *KI is reset, the MiniDOS filter will re-use its initial memory allocation if activated again.

Once the MiniDOS filter is applied, pressing the <CLEAR><SHIFT> and the specified alphabetic key will cause the following:

<C> - Toggle the CLOCK display on or off.

<D> - Enter the system DEBUGger (if activated).

<F> - Display FREE space for all active drives.

<K> - Kill a file.

<Q> - Display a disk's directory.

<R> - Repeat the last DOS command.

<T> - Issue a Top Of Form to the lineprinter.

When the MiniDOS filter intercepts one of these keys, it will immediately execute the associated function. These keys are active inside any program that uses the LDOS keyboard driver, including LBASIC. When the function has been completed, control will be returned to the calling program as though no key had been pressed. For this reason, if some of these functions are executed from the DOS level, the LDOS Ready prompt will not appear on the screen when the operation is complete. However, the system is still positioned as if the prompt were on the screen, and is ready to take another input.

The full descriptions and parameters for each command will be listed here.

<C> - The <C> command will toggle the clock display on or off. This is identical to issuing a CLOCK (ON) or CLOCK (OFF) library command.

<D> - The <D> command will enter the system DEBUGger or extended DEBUGger, providing it has been previously activated with the DEBUG or DEBUG (EXT) command

<F> - The <F> command will allow you see the free space available on a specified drive, along with the disk's name and date of creation. After selecting this command, you will see the letter F enclosed within braces. At this point, enter the drive number of the target drive. The display will be in the following format:

NNNNNNNNDDDDDDDD FFFF K Free

N = the disk name.

D = the disk date of creation.

F = the free K (1024 bytes) in Hex notation.

<K> - The <K> command will allow you to kill a specified file. You will see the letter K appear with in braces. At this point, type in the filespec you wish to kill. If no drivespec is included, all drives will be searched and the first matching filespec will be killed.

<Q> - The <Q> command will show the visible files on a specified drive. You will see the letter Q appear within braces. Type in the drivespec of the target drive, and the visible files will be displayed in 4 across format. You may also specify a particular file extension. The syntax would be:

d/EXT

where "d" is the drivespec, and /EXT is the desired extension. The wildcard character ($) may be used, but all three characters must be specified. For example, to find all file extensions starting with B, /B$$ must be specified.

<R> - The <R> command will repeat the last issued DOS command.

<T> - The <T> command will issue a Top Of Form to the line printer. This will also clear the line counter.

Entering an invalid parameter for any of the above commands will display the associated LDOS error message.

P R I N T E R F I L T E R P R O G R A M (PR/FLT)

The FILTER program PR/FLT is provided to format the data sent to the line printer. The syntax is:

FILTER *PR PR/FLT (parm,parm,...)

parms are the parameters described below.

ADDLF Will add a linefeed after a carriage return.

CHARS The number of characters per printed line.

FFHARD Will issue an X'0C' for a form feed, rather than a series of linefeeds.

INDENT Number of characters to indent from left margin on lines longer than CHARS parm.

LINES The number of lines printed on each page.

MARGIN Sets the left margin.

PAGE Sets the physical page length in lines.

TAB Causes expansion of X'09' tab characters.

XLATE=X'aabb' specifies a one-character translation.

aa = the character to be translated.
bb = what "aa" will be translated to.

ZERO Causes the printer line counter to start at 0, rather than the normal 1.

abbr: All parameters can be abbreviated by their first character.

PR/FLT is used to FILTER the devspec *PR. For a complete explanation of the FILTER LIBrary command, please refer to that section.

This FILTER program adds certain enhancements to the normal ROM printer driver routine. If you have entered a command that uses the printer, you will no longer experience "lock up" if the printer is not connected to the system. Realize that if the printer is merely in a deselected or alert state, the system will wait until printer capabilities have been re-established.

Once PR/FLT has been applied, *PR may be returned to its power up driver with the RESET *PR library command. If you wish to re-apply PR/FLT, it will occupy the same high memory initially allocated.

This FILTER program also adds two features to operation under LBASIC. The command LPRINT CHR$(6) will reset the system line counter to top of page. This may be used when manually positioning to top of form. Also, the command LPRINT with no arguments will now cause a blank line to be generated,

****** NOTE ******

Do not use a CHR$(138) as a linefeed character from LBASIC. This character is not intended to be used as a linefeed, and will cause printer operation problems.

The PR/FLT filter will allow you to determine the format of the data sent to your line printer. There are 9 configurable parameters used to set the format of the PR/FLT output. They are:

ADDLF If this parameter is specified, a linefeed will be issued after every carriage return.

CHARS= This parameter sets the number of characters that will be printed on each line. It may be any number between 1 and 255.

FFHARD If this parameter is specified, any form feed determined by the PAGE and LINES parameters will be sent as an X'0C' character rather than a series of linefeeds. If you use this parameter, be sure your printer will recognize the X'0C' character.

INDENT= This parameter sets the number of spaces a line is to be INDENTed if the line length exceeds (CHARS=) characters. The default value for this parameter is zero (0).

LINES= This parameter sets the number of lines that will be printed on each page. It may not exceed the PAGE parameter, and if not specified, it will default to the PAGE parameter of 66.

MARGIN= This parameter sets the width of the left margin. It is especially useful for printers with fixed position tractors.

PAGE= This parameter sets the physical page size in lines. It should be set to the particular form size you are printing on (66 for normal printer paper, 6 for mailing labels, etc.). The default value is 66 lines per page.

TAB If this parameter is specified, any X'09' character will be expanded to a standard 8 column tab.

XLATE This parameter will translate a specified character to another character. The format is X'aabb', where aa is the character to be translated, and bb is desired character result. This parameter may be useful to translate printer control characters when using more than one type of printer on the same system.

ZERO This parameter will cause the printer line counter to start from 0, rather than 1, as is the normal default. This may be necessary for compatibility with existing software.

```
FILTER *PR USING PR/FLT (CHARS=80,INDENT=6,PAGE=51,LINES=45,FFHARD)
```

This command will establish the PR/FLT program in high memory and FILTER the following parameters for the *PR (line printer) output.

(CHARS=80) will allow a maximum of 80 characters per printed line. If a line contains more than 80 characters, the excess will be printed on the next line(s).

(INDENT=6) will indent 6 spaces the remainder(s) of any line that exceeds 80 characters (determined by the CHARS=80 parameter).

(PAGE=51) sets the physical page size to 51 lines.

(LINES=45) will allow for 45 lines to be printed on a page. Since the page length is 51 lines (determined by the PAGE=51 parameter), the PR/FLT program will normally send 6 linefeeds after the 45th line has been printed. These linefeeds are determined by the formula (PAGE minus LINES). If no linefeeds are required, do not specify either PAGE or LINES. NOTE: Since the FFHARD parameter was used, an X'0C' (top of form) character will be sent instead of the 6 linefeeds.

(FFHARD) will cause an X'0C' to be sent rather than 6 linefeeds when the line count reaches 45.

```
FILTER *PR USING PR/FLT (MARGIN=10,CHARS=80,INDENT=6)
FILTER *PR PR (M=10,C=80,I=6)
```

This example will cause all lines to start 10 spaces in from the normal left-hand starting position (MARGIN=10). Any line longer than 80 characters will be indented 6 spaces when wrapped around.

```
FILTER *PR PR (TAB,ADDLF)
FILTER *PR PR (T,A)
```

This example will cause expansion of all X'09' characters to their normal 8 column tab position. Also, a linefeed will be sent every time a carriage return is sent.

```
FILTER *PR PR (XLATE=X'2A2E')
FILTER *PR PR (X=X'2A2E')
```

This example will translate all X'2A' characters (asterisks) to an X'2E' characters (periods). This may be useful to change the appearance of a report format.

```
FILTER *PR PR (FFHARD,ZERO)
FILTER *PR PR (F,Z)
```

This example will issue a Top of Form as an X'0C' Top of Form character, not as a series of linefeeds. Additionally, the printer line counter will start from 0 rather than 1, as the ZERO parameter was specified.

J O B C O N T R O L L A N G U A G E (JCL)

The Job Control Language is one of the most powerful features of the LDOS Disk Operating System. JCL allows the user to compile a pre-established sequence of commands or other keystrokes stored in an ASCII file including LDOS commands, user program commands, and program query responses. LDOS provides for dynamic modification of the JCL at run time based on parameters passed to the procedure by the user. It will then automatically execute the sequence, requiring no additional input by the user.

Some restrictions on inclusion of LDOS library commands into JCL processing must be noted. First, the concept of JCL is to pre-establish a job stream for "hands-off" execution. That, in itself, precludes the running of jobs with unpredictable queries. Since it is absolutely essential that program queries be in synchronization with JCL file responses, any LDOS library command that could be capable of asking a question depending on variable run-time environment must be restricted from JCL entry. For example, the DEBUGger is a powerful system utility. However, it cannot be run under JCL, and DOing a JCL procedure will automatically turn off the DEBUGger. Also, no single line in a JCL can exceed 63 characters in length.

Several other system commands cannot be executed from within a JCL file. The following lists those commands not acceptable for JCL:

```
COPY (x)
DEBUG
BUILD
PURGE
RESET *KI or RESET
SYSTEM (SYSGEN)
```

JCL can be constructed to run complete applications without operator intervention. It can also be used to minimize the command line entry of frequently used LDOS procedures. For instance, suppose your operation requires frequent use of the RS-232 driver. The command line you use to establish the RS-232 might be:

```
SET *CL TO RS232 (WORD=7,STOP=2,PARITY,EVEN,BAUD=450,DTR)
```

By BUILDing a single line JCL file with filespec "SETCL/JCL" containing the above sequence of characters, the Comm Line could be established by simply entering the command:

```
DO = SETCL
```

This reduces the keystrokes required to establish the Comm Line.

All programs that utilize the line input handler (identified as @KEYIN in the System Entry Point section) will be able to accept "keyboard" input from the JCL file, just as though you typed it in when the program was run. This gives the capability of pre-arranging the responses to a program's requests for input, inserting the responses into the JCL file, initiating the procedure,

then walking away from the machine while it goes about its business of running the entire job.

Keyboard input normally handled by the single-entry keyboard routines (@KBD, @KEY, and LBASIC's INKEY$) will continue to be requested from the keyboard at program run time and will not utilize the JCL file data for input requests. Thus by understanding fully the dynamics of JCL processing, you can write applications that take full advantage of the power inherent in the Job Control Language.

NOTE: ..... The LDOS command used to compile and/or execute a JCL file is the "DO" command. Instructions for its use may be found in the Library Command section of this manual.

A JCL file is processed in a two-phase procedure. In the first phase, called the compile phase, four basic operations will take place.

1) Various options available to dynamically modify the resultant JCL are examined.

2) User entered parameters are parsed and dealt with.

3) Token strings are operated on.

4) Conditional blocks of JCL are either utilized or not depending on the logic of token operators.

If all parameters in your JCL file are constant (such as the "SETCL/JCL" discussed earlier), no compilation is necessary. The DO command includes a method of forcing the JCL processor to completely skip the compile phase. This method is explained in the "DO" Library Command section. Bear in mind, that if your JCL contains language that requires compilation, skipping the compile phase will most likely result in failure of the JCL execution. At best, the JCL execution phase will abort.

In the execution phase, the LDOS JCL execution overlay (SYS11/SYS) will be called upon to carry out your commands. This phase also provides additional capabilities and functions. Of particular importance is the execution phase conditional blocks which can be used to provide run time menus or other run time variations of the JCL. This phase also provides for flashing prompts, time delays, and alerting tones.

JCL USED WITH LBASIC

One final note for the LBASIC users. All JCL capabilities available at the DOS level are equally as available at the LBASIC level, excluding library commands that alter high memory. A running LBASIC program can initiate a JCL procedure by using the following format in a program line.

```
CMD"DO filespec (parm1,parm2,...)
```

The JCL will compile, execute, and return to your LBASIC program to continue.

J C L C O M P I L A T I O N A N D P R O C E S S I N G

During compilation, the JCL file specified in the "DO" command line is examined line by line. If parameters are passed in the command line, they will be acted upon. Additionally, other conditional operations are dealt with and token substitution is performed as required. The resultant job stream is written to a file called "SYSTEM/JCL". Control would normally then be passed to the second phase - the execution of the compiled SYSTEM/JCL file.

NOTE: ..... The DO Library Command section will detail how to compile a JCL file without execution. This is a useful function to provide for the complete testing and examination of your logic in the compile phase prior to execution of the JCL.

The compile phase can also implement a labeling procedure. A series of JCL files can be melded into one large JCL file by providing an alphanumeric label for each distinctly separate procedure (for ease of explanation, we will refer to this file as PROCLIB - PROCedures LIBrary). This has the advantage of not wasting valuable disk space for one and two line JCL files. For that matter, even larger JCL procedures can be part of an entire PROCLIB. The only limit is your disk space availability. The labeled procedure to be compiled during the compile phase would be selected by passing its label in the parameter field of the "DO" command line.

The parameter field is used to pass information to the compile phase. Acceptable parameters will depend on the actual contents of the JCL file. In general) the parameter field may contain tokens, values for the tokens, and labels.

A token is similar to a variable in LBASIC, except that it can be up to eight significant characters in length (all eight significant), and can contain the characters A-Z, a-z, and 0-9. Upper case and lower case letters are treated the same. That is, the letter "A" and the letter "a" are equivalent. JCL does not distinguish between upper or lower case, but treats both as upper case.

The token values can be up to 32 characters in length. The values also can be in either upper case or lower case. The character set accepted is A-Z, a-z, 0-9, as well as the three special characters - slash (/), period (.), and colon (:). This character set will be sufficient for most purposes.

A label may be identified by an AT sign (@) prefix, which identifies it as a label rather than a token. A label may contain up to eight alphanumeric characters. For example, let's assume that the command line is entered as:

```
DO PROCLIB (@GAMES,LBASIC,GAME=CHECKERS)
```

The procedure "@GAMES" would be selected from the PROCLIB file, a token "LBASIC" would be created, and a token "GAME" would be established with the string value "CHECKERS". You can correlate this action to BASIC programming where "LBASIC" would be a null string and "GAME" would be a string of the eight characters "CHECKERS".

If the DO command line will exceed 64 characters, refer to the DO Library Command section for information on how to extend the DO command line.

Within the JCL itself, two special field types exist. These are the string substitution field denoted by a token placed within pound signs "#" (#GAME#, #NAME#, and #FILESPEC# are all string substitution fields), and an expression consisting of one or more tokens with logical operators placed between them. During the compile phase, any string substitution tokens found in the JCL file being processed are replaced with their current string value - either assigned in the command line (as in GAME=CHECKERS) or in the "//ASSIGN" JCL macro identified later. For example, suppose a DO command line included the parameter:

```
GAME=CHECKERS
```

Wherever #GAME# appeared in the JCL, it would be replaced by the character string, "CHECKERS". This is an extremely useful and powerful capability.

Expression fields are also evaluated during the compile phase. Any token passed in the DO command line parameter field is automatically declared to have a logical value of "TRUE", regardless of its "string" value. Tokens can also be declared logically TRUE or logically FALSE by means of the //SET and //RESET JCL macros. These token expressions can contain a mixture of three distinct logic operators. The operators are:

```
logical NOT ..... identified by a dash "-"
logical AND ..... identified by an ampersand "&"
logical OR  ..... identified by a plus sign "+"
```

The processing is strictly left to right. Parentheses do not change left to right evaluation. All operators have the same precedence.

Since the above remarks have been general in nature, it is recommended that you examine in depth the documentation on each JCL macro, as well as all example JCL procedures. It is also recommended that experimentation be done with non-essential files prior to implementing JCL fully.

JCL COMPILATION MACRO'S (VERBS)

The following Job Control Language macros are evaluated and processed during the compile phase of JCL. If your procedures contain any of these macros, it is essential that you perform the compile phase for proper results.

@LABEL

This defines the beginning of a Job Control Language procedure. The procedure will start from the line immediately following the "@LABEL", and will extend to but not include any subsequent @LABEL found in the processed stream. The procedure will also conclude if the end of the JCL file is reached before any other label is found.

It is not necessary for a JCL file to contain an @LABEL. Any JCL file without an @LABEL is considered to be a single procedure. If the JCL file does contain labeled procedures, it is necessary for the command line to designate an @LABEL in the parameter field. The labels are not written to the SYSTEM/JCL file and the JCL execution overlay cannot process a label line. If you attempt to skip the compilation of a labeled procedure, the execution phase will abort.

It is possible to combine labeled procedures in JCL conditional blocks. For example, consider the following brief JCL PROCLIB:

```
. Master procedure library - Version 5.0
@FIRST
//. Compiling the @FIRST procedure - please standby...
. example procedure to concatenate labeled procedures
. This is the first procedure
//IF COMBINE
DO PROCLIB (@SECOND)
//END
@SECOND
//. Compiling the @SECOND procedure ...
. this is a second phase of the total procedure
. This is the second procedure
```

Two procedures are identified in this PROCLIB. These are labeled "@FIRST" and "@SECOND". If the DO command line is entered as:

```
DO PROCLIB (@FIRST)
```

only the JCL from @FIRST to @SECOND is compiled, since the token "COMBINE" is not passed as a parameter and is considered to take on the logic state of FALSE. Thus, the //IF macro conditional will be FALSE. If the DO command line is entered as:

```
DO PROCLIB (@SECOND)
```

only the second procedure will be compiled. However, if the command line is entered as:

```
DO PROCLIB (@FIRST,COMBINE) or DO PROCLIB (COMBINE,@FIRST)
```

then the procedure labeled "@FIRST" would be compiled with the LDOS command "DO PROCLIB (@SECOND)" as part of its job stream. At the conclusion of the @FIRST procedure, the PROCLIB will automatically be re-compiled by the "DO PROCLIB (@SECOND)" request. This demonstrates the flexibility of JCL procedures "DOing" other JCL procedures. This power is also available from LBASIC. LBASIC can implement the command:

```
CMD"DO PROCLIB (@FIRST,COMBINE)
```

as part of a running program.

//. <COMPILATION COMMENT STRING>

This macro is used to display a comment line during the compile phase. The message is not placed in the SYSTEM/JCL file. Status messages are not normally displayed to the screen during the compilation phase except if the job aborts. These compile comment strings will inform you of the current compile status. The compile comments may require a space following the //. (the space is not required if the word that follows the //. is NOT a MACRO, TOKEN or @LABEL).

//INCLUDE <filespec>

This macro can be used to merge multiple files into the compilation job stream. It is useful when writing large and complex JCL procedures. The procedures can be written as separate compilable files for logical and operational testing. By using the //INCLUDE macro, the individual files can be effectively brought together.

It is possible for //INCLUDEd JCL files to //INCLUDE other JCL files. This is called nesting (similar to a nested FOR-NEXT loop in LBASIC). The maximum nest level is ten. That is, only ten //INCLUDEs can be active at any one time. For example:

```
//. NEST0/JCL
. nested procedure example
//INCLUDE nest1
/ this is the end of the primary JCL
//EXIT

//. NEST1/JCL
. this is the first nest
//INCLUDE nest2
. this is the end of the first nest

//. NEST2/JCL
. this is the second nest
```

The above will result in a nest level of two (two pending //INCLUDEs). If these three JCL files are saved as NEST0/JCL, NEST1/JCL, and NEST2/JCL, and the NEST0/JCL is compiled and executed) it will result in the following dialogue:

```
//. NEST0/JCL
//. NEST1/JCL
//. NEST2/JCL
. nested procedure example
. this is the first nest
. this is the second nest
. this is the end of the first nest
. this is the end of the primary JCL
```

Realize that an //INCLUDE macro cannot end a JCL file. If it does (for instance, if you leave off the last line of the NEST1/JCL in the above example), you will see the error message "RECORD NUMBER OUT OF RANGE"

//IF <expression>

This macro initiates a compilation phase conditional block. The expression can contain tokens and/or logic operators. The resultant logic state will determine the inclusion of the JCL between the //IF and matching //END macro. The limit of tokens and operators that can be included in the expression is the amount that can fit on one line. Consider the following JCL file named LOGIC/JCL.

```
. LOGIC/JCL
//.  compiling logic operator example...
. this comment line will be written to the SYSTEM/JCL file
//IF rel&lst
. "rel" and "lst" were both passed as tokens
//ELSE
//IF rel
. only "rel" was passed as a token
//ELSE
//IF lst
. only  "lst" was passed as a token
//ELSE
. neither "rel" nor "lst" was passed as a token
//END
//END
//END
//IF rel+lst
. either "rel" or "lst" or maybe both were passed as tokens
//END
//IF -rel
. "rel" was not passed as a token
//END
//IF -lst
. "lst" was not passed as a token
//END
//EXIT
```

Please study this example fully. There is much logic contained in it. All three operators are used to generate various expressions. Just about all combinations of logic states are explored. After you think you understand what is going on, try the four following example DO commands:

```
DO LOGIC
DO LOGIC (rel)
DO LOGIC (lst)
DO LOGIC (rel,lst) or DO LOGIC (lst,rel)
```

//ELSE

The //ELSE macro, when used after an //IF macro, specifies an alternative course of action in case the logical result of the preceding //IF test is FALSE. The example provided with the //IF macro should be sufficient for proper understanding of this macro.

//END

The FOR-NEXT loop in BASIC can be easily correlated with the //IF-//END in JCL. If you are using nested FOR-NEXT loops, then each FOR must have a corresponding NEXT. Such is the case with the //IF macro. The end of the //IF JCL is denoted by the //END macro. Each and every //IF must have a corresponding //END. Inspect the LOGIC/JCL example included with the //IF macro for a proper syntax of the //END placement within the JCL.

//SET <token>

A token is defined to be a logic TRUE if it is passed in the parameter field of the DO command line. Another way of defining it to be a logic TRUE is by using the //SET macro. For example, you may set a series of tokens TRUE, depending on the presence of a specified token in the parameter field.

```
TFTEST/JCL
//. this is compiling
//IF doit
//SET one
//SET two
//SET dontdoit
//END
//IF dontdoit
. either doit or dontdoit
//END
```

In this example, passing the token "doit" in the parameter field will result in the first conditional being set to a logic state of TRUE. The three //SET macros will then be compiled. The third, //SET dontdoit, will assign a logic TRUE to the token, "dontdoit". Thus, the second conditional //IF macro will reflect a logic TRUE and the comment line will be compiled to SYSTEM/JCL. Obviously, the comment line could be replaced by a very large JCL procedure.

If "dontdoit" was passed as a token, it would receive a logic state of TRUE. However, since the token "doit" would be a logic FALSE, the //SET macros would not be compiled.

//RESET <token>

The //RESET macro performs the exact opposite of the //SET macro. A token will take on a logic FALSE if it is //RESET.

//ASSIGN <token>=<string of characters>

The ASSIGNment macro performs essentially the same thing within a JCL procedure as declaring a token value in the parameter field. The command statement:

```
DO PROCLIB (@ASSIGN,BIGNAME=desired value)
```

achieves exactly the same result as:

```
@ASSIGN
. this macro does what the command line did - but...
. it can not be altered at run time!
//ASSIGN BIGNAME=desired value
```

The difference is that the ASSIGNment within the JCL procedure can only be altered by editing the JCL file, whereas, in the parameter line ASSIGNment, it can be different each time the JCL is compiled Nevertheless, ASSIGNments can play an important part in the overall utility of JCL procedures.

//QUIT

This macro will be useful when you want to abort the compilation of a JCL procedure. It will abort the processing without going to the execution phase. Consider a situation where the JCL requires a parameter token. If the user does not supply such a token in the DO command line, the JCL should abort. Try this example:

```
. Q/JCL
//. compiling...
. this JCL procedure requires a "name" token assignment
//IF -name
//. you did not supply a name!!! I must abort!!!
//QUIT
//END
LIST #name# (hex)
```

If you "DO Q" (without passing the parameter NAME), the LIST command will not be performed. This is useful because the LDOS command LIST requires a filespec. It should be noted that if the token NAME is passed, it must be assigned a value which represents a legitimate filespec. If the token NAME was passed and not assigned a value, the error message FILE SPEC REQUIRED will appear.

TOKEN CONCATENATION

One final topic to be included in the compilation phase is that of string concatenation. Whenever a string value is being compiled, any adjacent string values would be concatenated to form a longer string. The following examples of string concatenation assume that the token #a# has the string value "MYFILE", #b# has the string value "ASM", and #c# has the string value "OBJ".

```
. testing string substitution and concatenation
LIST #a#/#b#.secret:2 (TAB,NUM,P)
LIST #a#/#c#:2 (HEX,P)
```

After compilation, this example would appear as.

```
. testing string substitution and concatenation
LIST MYFILE/ASM.SECRET:2 (TAB,NUM,P)
LIST MYFILE/OBJ:2 (HEX,P)
```

NOTE: Since the pound sign (#) is used to mark off string substitution fields) a compiled JCL procedure cannot contain a single pound sign. If a pound sign is needed, use two consecutive pound signs (##). This will be translated by the compiler into a single symbol.

J C L E X E C U T I O N

The execution phase of JCL processing monitors the ongoing interface between the operating system's activities and the instructions contained in the JCL file being executed. One requirement of executing a JCL file is that its first line must be a comment, an LDOS library command, or an executable program command line. The first line cannot be an execution phase macro. The following comment strings and macros are processed during the execution phase.

. <comment message string>

Any line beginning with a period is considered to be a comment and will not be included in execution. A comment line will be displayed to the screen for informational purposes. If you have an active job log in effect, the message will also be time-stamped and sent to the job log device (*JL). A comment is most useful as a tag identifying the JCL file name. If used as the first line, you will avoid complications that result if an execution macro comes first. There should be a single space between the period and the comment string.

//EXIT

The //EXIT macro is used to exit the execution of JCL processing and return to the LDOS Ready prompt, displaying the message:

```
JOB DONE
```

This type of JCL exit should be used if the conclusion of the JCL command file also represents the conclusion of the job that is running. If the application is to continue, then use the //STOP macro. For example, suppose you generated the following 3-line JCL file:

```
SET *CL RS232 (WORD=8,STOP=1,PARITY=NO,DTR)
LCOMM *CL (XLATE=X'3B5F')
//EXIT
```

When you would DO this JCL, it would function until the first file prompt request was performed in LCOMM. Then, since //EXIT would be executed, a return to LDOS would take effect - not quite what you would want to happen. The //EXIT macro should only be used if your job is to terminate.

//ABORT

The //ABORT macro is quite similar to the //EXIT macro. A return to LDOS Ready will take effect after displaying the message:

```
JOB ABORTED
```

It would be used if your JCL processing logic detected an invalid run-time condition, and wanted to display a different informative message. Also, any error that the operating system detects that will result in a jump to the @ABORT DOS routine will disable further JCL processing and display the above message.

//STOP

The //STOP macro is used to cease further JCL execution and return to the application originating the @KEYIN request. Since control is returned to the keyboard, no JCL message is displayed to the screen. Any prompting for input must come from the running application. If the JCL file used as an example in the //EXIT discussion was constructed as:

```
SET *CL RS232 (WORD=8,STOP=1,PARITY=NO,DTR)
LCOMM *CL (XLATE=X'3B5F')
//STOP
```

then as soon as the first file prompt was entered, JCL execution would cease, and the keyboard would become "alive". The response to the prompt would have to come from the keyboard.

Bear in mind, if you close out a job with the //STOP macro and the job has just concluded, then the system may appear to "hang" as no prompt for keyboard input would be given. Although the desire was to return to LDOS Ready, the //STOP keyboard request would come prior to any LDOS Ready prompt and you may not realize it. Make sure you use the proper exit macro for the intended job application.

//PAUSE <comment message string>

This macro will temporarily suspend program execution. The entire job will go into a wait state. The resumption of the job will take place upon depressing the <ENTER> key. Pressing the <BREAK> key will abort the job.

```
. this JCL will pause
clock (on)
armalarm (door=3,window-5,phone="7035551212")
//pause After pressing ENTER, you have 25 seconds to leave
//delay 250
create joblog:2 (s=5)
route *jl joblog:2!
lbasic (blk=50) RUN"security/bas"
//stop
```

This JCL could initiate a procedure to start up your home security system. It will arm the various alarms, set up the automatic telephone calling unit, wait for your ENTER, then start up the BASIC security program. Of course, you must supply the appropriate software and hardware interfacing.

However, the JCL would do all the rest. Note the use of the exclamation point suffix to the filespec. It forces the directory to be constantly updated as the job log file is written. This is explained in the Glossary (see filespec). Notice also that this example is in lower case. This is perfectly acceptable, as lower case may be intermixed with upper case as desired.

//ALERT (<tone>,<silence>,...)

The //ALERT macro may be used to provide an audible signal to the operator. It will generate up to eight different tones and direct its output to the cassette port. By plugging a small amplifier into the cassette port, this macro could prove quite useful. You could use it to signify the end of a large JCL procedure. It could also be used during the execution of a procedure to bring attention to a specific process.

The actual tone selected is controlled by a tone number. The number range is 0-7, with "7" producing the lowest tone, and "0" producing the highest tone. Any value entered will be used in its modulo 8 form. That is, if you enter the number "8", a zero value will be assumed. The value 65 will produce the tone assigned to a "1". The tone is followed by a period of silence by entering a second number. Tone and silence must be entered as number pairs (e.g. "1,0"). In fact, this can be repeated for as many number combinations as can fit on one line.

The tone-silence "string" can be made to repeat by enclosing the entire string in parentheses. If parentheses are used, the string will keep repeating until the <ENTER> key is pressed. No display is made during the tone generation. Therefore, if your JCL has a repeating tone and you do not have an amplifier connected to the cassette port, the system may appear to hang.

Try the following JCL procedure:

```
. example of tone generation
//alert (1,0,7,0)
. another example
//alert 0,0,1,0,2,0,3,0,4,0,5,0,6,0,7,0
. still another
//alert (0,0,1,1,2,2,3,3,4,4,5,5,6,6,7,7)
//exit
```

//FLASH <duration> <message string>

The //FLASH macro is used to blink a one-line message on the video display. The "duration" is optional. If omitted, a duration of 256 blinks will be used. If the <ENTER> key is pressed during the flashing of a message line, the flashing will immediately cease and the next JCL line will be fetched.

```
. BU/JCL - sample BACKUP procedure requiring compilation
//.   Compilation of sample BACKUP procedure underway
//if -d
//assign d=7
//end
//flash 10 The BACKUP operation is starting...
//pause Load destination disk and <ENTER> to resume
backup /asm:1 to #d# (mod)
//alert (1,0,7,0)
//exit
```

Using this procedure, a simple "DO BU (d=4)" will carry out the procedure. Note the use of the conditional block to assign a default destination drive if it is omitted in the parameter line. The //FLASH is used to draw attention to the next request, which is a //PAUSE. After the BACKUP has been made, a repeating siren sound will be heard until stopped with the <ENTER> key.

//DELAY <duration>

The //DELAY macro will provide for a definite timed pause. Execution will automatically continue at the expiration of the delay period. The actual delay will be approximately 0.1 seconds per count. The count may range from 1 to 256. Thus, a delay from 0.1 seconds to a delay of 25.6 seconds is possible. Similar to the tone generation, the value used will be modulo 256. An example of the use of //DELAY was demonstrated with the JCL procedure provided as a //PAUSE example.

//WAIT hh:mm:ss

This macro can be quite useful in an application where certain jobs must be run at a specific time period. Providing the system clock is functioning, the //WAIT macro will put the entire system in a "sleep" state until such time as the system clock matches the time specified in the macro operand. During the time that the system is asleep, it will not be disturbed.

```
. example nighttime job scheduler
//. compiling night jobs
//if -job
//assign job=standard/job
//end
system (update)
clock (on)
//include setcl
//wait 02:15:00
lbasic
run "#job#"
```

This JCL procedure includes a number of the various macros already discussed. It demonstrates default filespec assignments, inclusion of other JCL procedures, useful LDOS library commands, and the running of a

BASIC job at a predetermined time period. JCL is extremely flexible and useful. You may even find yourself building entire application systems around the JCL constructs.

JCL EXECUTION CONDITIONALS

The next three macros are used together to establish execution-time conditional blocks. These are blocks of JCL that are selected during the execution phase of the JCL procedure. A common use would be the display of a menu containing various options that the operator could select.

//KEYIN <comment string>

This macro is used to prompt for a single character entry. This entry will be used to select one of up to ten different execution phase conditional blocks of JCL. The entire //KEYIN line will be displayed, including any comment message. The JCL will then accept a single character from the keyboard. This character must be in the range 0-9. It will be used to search for a conditional block tagged with the same label character. //KEYIN can not be used to enter data at execution time but can only provide for the selection of one or more alternatives. If it is necessary to provide run-time keyboard interfacing, then the //INPUT macro should be used instead.

//<character>

This macro is used to label an execution phase conditional block. Its use is associated with the //KEYIN macro. The character "tag" can be in the range 0-9 and will be the character matched in the //KEYIN search procedure.

///

The triple-slash is used to denote the conclusion of ALL execution phase conditional blocks. A single conditional block extends from its character tag label to the next conditional block's character tag label or until the triple slash is detected. Do not forget to incorporate it into your JCL file if you are going to use execution phase conditionals.

The following JCL procedure should demonstrate the use of execution phase conditionals:

```
. in case of reboot...
auto do demo
. Example of a menu selection procedure
//. Compilation of LDOS Demonstration
//.
. Welcome to the LDOS Operating System
.
. What would you like me to demonstrate?
. 1> Operation of LBASIC?
. 2> A sampler of library commands?
. 3> Or perhaps the capabilities of the LX-80 interface?
.
//keyin your selection (1-3)
//1
LBASIC RUN"DEMO/BAS"
CMD"S
DO *
//exit
//2
dir :0 (a,s,i)
date
clock (on)
device
lib
do *
//exit
//3
LX80DEMO (hard,eight,five,sio)
do *
//exit
///
.
. Well! I see you did not like my choices.
BOOT
```

//INPUT <message string>

This macro could very well turn out to be one of the most powerful execution phase macros. It provides a means of forcing an input to be fetched from the keyboard, rather than from the JCL file. When the //INPUT macro is detected and executed, it will re-establish the keyboard as the input device (unless some other device was originally linked to the keyboard). Keyboard input will continue until either a <BREAK> is detected or the <ENTER> key is pressed. When either of the two conditions prevail, input fetching will revert to the JCL file. The <ENTER> key will satisfy the pending @KEYIN or LBASIC INPUT statement. The <BREAK> key will cause an ABORT condition and the executing job will terminate.

Consider this rewrite of the menu example previously discussed.

```
. in case of reboot...
auto do demo
. Example of a menu selection procedure
//. Compilation of LDOS Demonstration
//.
. Welcome to the LDOS Operating System
.
. What would you like me to demonstrate?
. 1> Operation of LBASIC?
. 2> A sampler of library commands?
. 3> Or perhaps the capabilities of the LX-80 interface?
.
//keyin your selection (1-3)
//1
LBASIC RUN"DEMO/BAS"
CMD"S
DO *
//exit
//2
dir :0 (a,s,i)
date
clock (on)
device
do *
//exit
//3
LX80DEMO (hard,eight,five,sio)
do *
//exit
///
.
. Well! I see you did not like my choices.
lib
//input your own choice from the LDOS library!
do *
//exit
```

If the operator does not choose a selection from the menu provided, a personal selection can be made. The //INPUT could also be used during the execution of a program to satisfy some input request from the keyboard instead of the JCL file if the application warrants it. The //INPUT could also be part of a compile phase conditional block that would appear in the compiled JCL depending on certain other conditions.

This concludes our discussion of JCL. Following will be usable examples of the JCL language. Feel free to type these examples in and experiment.

J C L P R A C T I C A L E X A M P L E S

This example demonstrates the way to set up a FORMAT selection JCL file (named GEN/JCL). It assumes that you have 40 cylinder drives, and can FORMAT in double density. The first two labeled functions will set the disk Name to "LDOS-5.1". The last two labeled functions allow you to pass the disk Name from the DO command line, using the token "NA". These examples can easily be adapted to your specific disk drive requirements. The disk Master Password, density, step rate, and number of cylinders can be set as described in the FORMAT Utility section.

```
//.generalized DO file GEN/JCL
//. ALL DRIVE 0 FUNCTIONS MUST BE ENTERED DIRECTLY !!
//abort
@FOR1D
//. compiling an LDOS system disk format, drive 1.
FORMAT :1 (DDEN,STEP=0,NAME="LDOS-5.1",MPW="PASSWORD",ABS)
//EXIT
@FOR2D
//.  compiling an LDOS system disk format, drive 2.
FORMAT :2 (DDEN,STEP=0,NAME="LDOS-5.1",MPW="PASSWORD",ABS)
//EXIT
@FOR1DN
//IF NA
//. FORMAT on drive 1, your disk Name is ==>  #NA#
//. Master disk Password is ==>  PASSWORD
FORMAT :1 (DDEN,STEP=0,MPW="PASSWORD",ABS,NAME="#NA#")
//EXIT
//END
. MISSING DISK NAME !!!  ENTER AS  NA=
//ABORT
@FOR2DN
//IF NA
//. FORMAT on drive 2, your disk name is ==>  #NA#
//. Master disk Password is ==>  PASSWORD
FORMAT :2 (DDEN,STEP=0,MPW="PASSWORD",ABS,NAME="#NA#")
//EXIT
//END
. MISSING DISK NAME !!!  ENTER AS   NA=
//ABORT
```

The first two labeled procedures set the FORMAT parameters and use the disk Name "LDOS-5.1" (see the FORMAT Utility section for default parameter values). The last two procedures check to see if a disk Name was passed with the NA token in the DO command line. If NA was not passed, the //IF conditional will not be satisfied and the message "MISSING DISK NAME !!!" will appear on the screen, and the DO will abort. An example of passing a disk Name might be:

```
DO GEN (@FOR2DN,NA=DATA2)
```

This command will start the DO execution with the label @FOR2DN. The screen will show the disk Name passed with the token NA and the Master Password as PASSWORD. You would see the FORMAT command line appear on the screen, and then the FORMAT Utility program would load and execute.

The parameters specified in the @FOR2DN line either satisfy all possible conditions required by the FORMAT Utility or assume default parameters. If the ABS parameter was not used, the DO would abort if the disk had been previously formatted. The SIDES and CYLinders are not specified and will default to one SIDE and 40 CYLinders.

To enter LBASIC and run a program, refer to the following example:

```
LBASIC RUN"program name"
//STOP
```

The //STOP macro is absolutely necessary to prevent an exit back to the LDOS Ready prompt.

Another practical use for the JCL feature is to use it as an alert function from LBASIC to notify you when a long, unattended process is complete. Refer to the following example of a JCL file called ALERT/JCL:

```
.Your process has completed
.
.Please press <ENTER> to continue your program
//alert (1,0,7,0)
//exit
```

If an appropriate amplifier were hooked to the cassette port, you would hear an alternating tone when this JCL executed. Note the //exit macro ending the JCL. This is necessary if control is to be returned to the LBASIC program.

You could call this JCL from LBASIC with a program line such as:

```
100 CMD"DO ALERT/JCL"
```

L B A S I C

This is the syntax to be observed when entering LBASIC.

LBASIC (parm,parm,...,parm) command

LBASIC * used to re-enter LBASIC with the program and the variables intact.

The allowable parameters are as follows:

BLK= parameter that specifies Blocked file mode, either ON or OFF. ON is the default.

FILES= parameter that specifies the maximum number of files LBASIC will be able to access (1 to 15). If not specified, 3 is assumed.

MEM= parameter to set the highest memory address to be used by LBASIC. All memory above this address will be "protected". If not specified, all memory up to HIGH$ will be available.

HIGH Parameter that sets the cassette baud rate,
LOW either HIGH or LOW (HIGH=1500 and LOW=500). The default is HIGH.

command This may be any valid LBASIC command which will execute immediately upon entering LBASIC, such as RUN"MYPROG/BAS", AUT0100, etc.

abbr: BLK=B, FILES=F, MEM=M, ON=Y, OFF=N, HIGH=H, LOW=L

LBASIC OVERLAYS

Three overlays are present on a Master LDOS diskette. Their functions are as follows:

LBASIC/OV1 - This overlay contains the renumbering program associated with the LBASIC CMD"N" function. It may be killed if no renumbering will be done.

LBASIC/OV2 - This overlay contains the cross reference program associated with the LBASIC CMD"X" function. It may be killed if no cross referencing will be done.

LBASIC/OV3 - This overlay contains the error handling routine used with LBASIC, along with the sort routine used for the CMD"O" function. It MUST be present when using LBASIC.

PROGRAM PROTECTION

LBASIC programs may be protected with an "Execute only" password. This means that the program may be RUN, but not LOADed, LISTed, LLISTed, or otherwise examined. Any attempt to break the program execution and examine the program will cause the program to be erased from memory, and the message "Protection has cleared memory" will be displayed. The DEBUGger will also be disabled during program execution.

NEW FILE CONTROLS

LBASIC provides a Blocked file mode (which has often been misnamed Variable Length Files). This mode allows files with Logical Record Lengths (LRL) of less than 256 bytes to be created and accessed. Any record length from 1 to 256 bytes will be allowed, even if the record size is not evenly divisible into 256. All blocking and de-blocking across "sector boundaries" will be performed by LDOS. In this way, user records can span across sectors to provide maximum disk storage capacity. If the LRL is not specified when OPENing a Random file, 256 will be assumed. Note that an LRL of 0 will signify a 256 byte LRL. Enhancements have also been made to the allowable methods of OPENing both Random and Sequential type files.

If the Blocked file mode is ON, each file declared when entering LBASIC will take 544 bytes of memory. If the Blocked mode is OFF, each file will take 288 bytes.

NEW RANDOM FILE CONTROLS

To specify a Blocked file, the file OPEN statement must be for a Random file. If you specified BLK=OFF when you entered LBASIC, the LRL of the Random files will be 256. Any of three Random modes are allowed:

RN	OPEN a New Random file (cannot already exist).
RO	OPEN an Old Random file (must already exist).
R	OPEN a Random file (whether or not it exists).

The proper OPEN command syntax is:

OPEN"random file mode",buffer number,"filespec",LRL

Examples of this type of OPEN are:

OPEN"RN",1,"DATA/NEW",89

This OPENs a random file named DATA/NEW, with an LRL of 89. If the file already exists, it will not be OPENed, and LBASIC will return a "File Already Exists" error.

OPEN"RO",1,"OLDDATA/DAT:0",34

This OPENs an existing random file. The file will be assumed to have an LRL of 34, whether or not it was created with that LRL. If the file does not exist, LBASIC will return a "File Not Found" error and will not create the file.

OPEN"R",1,"INDEX/DAT"

This statement will OPEN the file INDEX/DAT with an LRL of 256. If the file already exists, it will be OPENed with an LRL of 256 even if it was created with a different LRL. If the file does not exist, it will be created with an LRL of 256.

When using random files, the LOC function can be used to determine the current file position. The format for this function is:

LOC (n)

Where "n" is the file buffer number. The value returned is the number of the most recently accessed record. If the file was just opened, LOC will return the value 0.

NEW SEQUENTIAL FILE CONTROLS

The following enhancements have been added to the methods of OPENing Sequential files, to better determine their existing status, and to provide addition of data directly to the end of an existing file. All these enhancements pertain to the OPEN for Output mode (OPEN"O" and OPEN"E").

OPEN"E",1,"TEST/DAT"

This statement OPENs a new or existing file. The file will be positioned in such a manner that any subsequent PRINTs to the file will be to the END of this file. If this is a new file, this command will be identical to the normal OPEN"O" command.

OPEN"EN",1,"TEST/DAT"

This statement OPENs a new file and positions it to add to the end of this new file. If the file already exists, an error will occur. This command is identical to the OPEN"ON" command.

OPEN"EO",1,"TEST/DAT"

This statement OPENs an existing file and positions it to add to the end of the existing file. If the file does not exist, a "File not found" error will occur.

OPEN"O",1,"TEST/DAT"

This command OPENs a sequential file for output. It makes no difference if the file already exists or not. If the file does not exist it will be created.

OPEN"ON",1,"TEST/DAT"

This command OPENs a sequential file for Output ONLY if the file does not exist. If the file does exist , LBASIC will return a "File Already Exists" error.

OPEN"OO",1,"TEST/DAT"

This statement will OPEN a sequential file for Output, ONLY if the file already exists. The file will be positioned at its beginning, and all previous contents of this "existing" file will be destroyed. If the file does not exist, LBASIC will return a "File Not Found" error.

ABBREVIATED COMMANDS

Each of the following LBASIC commands may now be represented as single characters. When using a single character command, the effect will be identical to using the entire word. This abbreviated form is only acceptable when typed on a command line, not in a program line or JCL file.

A represents the command AUTO.

D represents the command DELETE.

E represents the command EDIT.

L represents the command LIST.

The following commands are implemented by pressing the indicated key as the first character in the command line. No carriage return is necessary; the indicated action will take place immediately.

. (period) This will perform the same function as the command "LIST.<ENTER>", which will instruct LBASIC to list the currently active line.

, (comma) This will perform the same function as the command ""EDIT.<ENTER>", which will instruct LBASIC to enter the "edit mode" for the currently active line.

<UP ARROW> This will cause LBASIC to display the next lower numbered line in the program.

<DOWN ARROW> This will cause LBASIC to display the next higher numbered line in the program.

<LEFT ARROW> This will cause LBASIC to display the first line of the program.

<RIGHT ARROW> This will cause LBASIC to display the last line of the program.

LBASIC SINGLE STEPPING

This new feature allows the LBASIC programmer to step through each program statement singly, with a "HOLD" after each step. To invoke this feature simply do a normal pause (<SHIFT @>), which will cause LBASIC to go into a wait state. While continuing to hold down the <SHIFT @> press the <SPACE BAR>, and the next LBASIC statement will execute. After execution of that statement the computer will immediately go into its wait state again. Holding down the <SPACE BAR> will execute statements at the normal keyboard repeat rate. If you press any key without holding down the <SHIFT @> then normal program execution will resume. Note that this feature also functions when listing a program.

SYSTEM COMMANDS (CMD)

LDOS LIBrary commands that do not affect HIGH$ may be executed from LBASIC. The syntax is CMD"command". For example:

```
CMD"DIR"
CMD"DEVICE"
CMD"LIST DAT1"
CMD"FREE"
```

All of the above will perform the LIBrary command contained within the quote marks and return to LBASIC. This type of CMD"command" will function whether called from LBASIC's "command line" or from within an LBASIC program. The command may also be contained in a string variable or expression, and called in the format:

```
CMD A$
```

Approximately 4K of free memory must be available, or an "OUT OF MEMORY" error will occur.

Several single letter CMD functions are also available. They are:

CMD"A"
Abnormal return to LDOS. Any active DO command will be cancelled.

CMD"B","switch"
This command will enable or disable the <BREAK> key, with switch being either ON or OFF. A string variable or expression may also be used.

CMD"D"
Turns on and enters the system DEBUGger.

CMD"D","switch"
The switch ON will turn on the system DEBUGger, but will remain in LBASIC. Hitting the <BREAK> key will enter the DEBUGger. The switch OFF will turn off the DEBUGger.

CMD"E"
Returns the last LDOS error message.

CMD"I","dos-command"
Exits LBASIC, passing a command to LDOS. You will not return to LBASIC.

CMD"L","filespec"
Loads a Load Module Format file into memory.

CMD"N"
The LBASIC renumbering function. Parameters are listed at the end of the LBASIC section.

CMD"O",number of elements to sort, first element of array to sort
This command will sort a single dimension string array. The sort will start at the element specified, and will sort the number of elements specified. The number of elements should not force the sort past the end of the array.

CMD"P",variable
This command will return the printer status in the variable specified. The variable may be any type, including a string. The value will have the bottom 4 bits stripped before being passed back to LBASIC.

CMD"R"
Turns on the clock display.

CMD"S"
Normal return to LDOS.

CMD"T"
Turns off the clock display.

CMD"X"
The LBASIC cross reference utility. Complete documentation can be found at the end of the LBASIC section.

NEW LBASIC COMMANDS

New commands have been added to LBASIC. They function as follows:

RESTORE nnnn

This command is similar to a regular RESTORE command, except that a line number may be specified. The data pointer will be moved to the beginning of the specified line. Any subsequent READ statement will start at the specified line when looking for DATA statements. The line need not contain any DATA statements, but the line number must be for an existing line. A variable may NOT be used in place of the line number.

RUN"filespec",V,line number

The V parameter of this command will allow saving all current variables and string space when running a new Basic program. A line number may be given to start execution of the new program at the specified point. The line number may NOT be specified as a variable. If no line number is given, the new program will begin execution at its first statement. An "Out of Memory" or "Out of String Space" error may occur, depending on the length of the new program, the number of active variables, and the amount of string space cleared. Be aware that DEF statements (DEFINT, DEFDBL, etc.) will not be carried to the new program.

RUN"filespec",line number

This command is similar to the above example, except that preservation of variables will not be done. The new program will begin execution at the specified line number.

SET EOFn

This command will allow a file opened in the "R", "RO", or ""RN" mode to have its End Of File (EOF) marker reset. The "n" is the buffer number of the file. The EOF will be set immediately after the most recently accessed record (written or read). For example:

```
GET 1,100:SET EOF1
```

will set the EOF of file 1 to record number 100. The command will be ignored if an attempted read past the current EOF was the last access of the file.

NEW ERROR NUMBERS AND MESSAGES

As a result of the Blocked file mode and the extended CMD functions, four new ERROR messages have been added to LBASIC.

FILE ALREADY EXISTS

This error will occur when using an OPEN"xN" command if the file already exists. The ERROR number is 116.

BLOCKED FILE ERROR

This error will occur if you attempt to OPEN a random file with an LRL of other than 256, after specifying BLK=OFF when entering LBASIC. The ERROR number is 140.

SYSTEM COMMAND ABORTED

This error will occur if an LDOS command called with the CMD"command" function aborts. The ERROR number is 142.

PROTECTION HAS CLEARED MEMORY

This error will occur if an attempt is made to illegally access an execute only program without using the proper password. The program and variables will be cleared from memory. The ERROR number is 144.

NOTE: The ERROR numbers are sometimes referred to in Radio Shack manuals as ERR/2 or ERR/2+1.

CMD"N" - LBASIC PROGRAM RENUMBERING
==========

This LBASIC feature will renumber LBASIC program line numbers as well as all line references such as GOSUB and GOTO. The syntax is:

CMD"N ! aaaa,bbbb,cccc,dddd"	
!	optional parameter to skip the complete scan for errors before renumbering begins.
aaaa	line number of the current program to start the renumbering from.
bbbb	new line number for line aaaa.
cccc	increment between line numbers.
dddd	the first line number above those to be renumbered minus one.

LBASIC/OV1 must be present on the disk, or a "Program Not Found" error will occur.

You cannot have a line number zero (0) if renumbering an LBASIC program.

This renumber feature will allow you to renumber all or parts of the LBASIC program currently in memory. The lines to be renumbered can be anywhere in the program. However, if the parameters you use would result in the renumbered lines being out of sequence, a BAD PARAMETERS error will occur.

If you do not specify the exclamation point (!) character, a full scan for errors will be done before the renumbering starts. If errors do exist, no lines will be changed. It is usually much easier to fix the errors before the lines are renumbered!

If you do specify the ! , any error found will still abort the renumbering. However, all internal line number references will have already been changed up to the line that cause the error! Do not use the ! parameter unless you are absolutely sure that no errors exist.

The default values for the line and increment parameters are as follows:

```
aaaa = 1
bbbb = 20
cccc = 20
dddd = 65529
```

CMD"X" - LBASIC CROSS REFERENCE
==========

This LBASIC feature will produce a cross reference of variables and line numbers for your LBASIC program currently in memory. The syntax is:

CMD"X devspec/filespec parameter,<title>"

devspec/filespec is the device or file the listing will be sent to. If not specified, it will go to the screen.

parameter specifies Variables or Lines as follows:

-V	all Variables.
=variable	only the variable specified.
-L	all Line numbers.
=number	only the line number specified.
<title>	an optional title to be printed on the top of each page.

LBASIC/OV2 must be present on a disk or a "Program Not Found" error will occur.

You cannot have a line number zero (0) if you wish to use the cross reference utility.

This cross reference feature will allow you to produce a list of the variable and line number references of an LBASIC program. This list may be sent to any device in the system, such as the *DO (video screen), *PR (line printer), etc. It may also be sent directly to a specified disk file.

Parameters are allowed to determine which variables or line numbers will be listed. If no parameter is specified, all variables and line numbers will be cross referenced.

If you wish a title to be put on the top of every page in the list, it must be specified between less-than/greater-than symbols in the command line.

TABLE OF CONTENTS - TECHNICAL SECTION

DCB - DEVICE CONTROL BLOCK

DCT - DRIVE CODE TABLE

DIRECTORY RECORDS

..... GRANULE ALLOCATION TABLE

..... HASH INDEX TABLE

DISK I/O TABLE

ENTRY POINTS

..... CONTROL

..... DISK PRIMITIVES

..... FILE CONTROL

..... GENERAL

..... ROM ENTRY

..... SPECIAL ROUTINES

ERROR DICTIONARY

FILE CONTROL BLOCK

FILE FORMATS

FILTERS AND DRIVERS

MEMORY MAP

RAM STORAGE AREAS

SUPERVISORY CALLS

SYSTEM OVERLAYS

D E V I C E C O N T R O L B L O C K

The Device Control Block (DCB) is used to interface with various logical devices such as the keyboard (*KI), the video display (*DO), a printer (*PR), a communications line (*CL), or other device defined by your hardware implementation. The DCB has three bytes of storage space available that may contain parameters associated with the specific device. For example, the current address position of the video cursor is contained in the video DCB. The maximum number of lines per page is stored in the Printer DCB.

The DCB follows a strict format that defines the utilization of the non-storage fields. This format must be followed in all Device Control Blocks established by the user. Each DCB is six characters in length. The following information provides specifications for each assigned DCB BYTE.

Byte 0 .... TYPE byte

Bit 7....This bit specifies that the Device Control Block is actually a File Control Block (FCB) with the file in an OPEN condition. Since there is a great deal of similarity between DCBs and FCBs, and devices may be routed to files, tracing a path through device links may reveal a "device" with this bit set, indicating a routing to a file.

Bit 6 & 5..These bits are reserved for future use.

Bit 4....If set, then the device defined by the DCB is ROUTEd to another device.

Bit 3....If set, then the device defined by the DCB is a NIL device. Any output directed to the device will be discarded. Any input request will be satisfied with a ZERO return condition.

Bit 2....If set, then the device defined by the DCB is capable of handling requests generated by the @CTL system call. See the System Entry Point Table for additional information.

Bit 1....If set, then the device defined by the DCB is capable of handling output requests which normally come from the @PUT system vector.

Bit 0....If set, then the device defined by the DCB is capable of handling requests for input which normally come from the @GET system vector.

Byte 1 .... Low-order Vector

This contains the low-order address of the driver routine that supports the hardware assigned to this DCB.

Byte 2 .... High-order Vector

This contains the high-order address of the driver routine that supports the hardware assigned to this DCB.

Bytes 3-5 .... Variable Storage Area

These three bytes are reserved for variable storage of parameters associated with each driver. It is up to the driver software to assign their use.

The system maintains space in low memory for the storage of the Device Control Blocks. This space is assigned as follows (note: address assignments are shown in hexadecimal):

Address Range	Assignment
4015 - 401C	*KI - Keyboard
401D - 4024	*DO - Video Display
4025 - 402C	*PR - Printer
42C2 - 42C7	*JL - Job Log
42C8 - 42CD	*SI - Standard Input
42CE - 42D3	*SO - Standard Output
42D4 - 42D9	- 1st Spare
42DA - 42DF	- 2nd Spare
42E0 - 42E5	- 3rd Spare
42E6 - 42EB	- 4th Spare

The 2 character device names will be stored in memory starting a X'42EC'. There is space for 10 device names, corresponding to the 10 device control blocks.

As can be observed, space for up to four additional devices, not currently defined in LDOS, has been made available. Additional devices may be defined, or existing devices redefined, by using the SET command coupled with an appropriate user-supplied DRIVER routine. Any device assigned by the user to a spare slot, may be removed from the system after the device is RESET by using the KILL devspec command. The LDOS defined devices are protected and cannot be killed.

D R I V E C O D E T A B L E (DCT)

The Drive Code Table (DCT) is the way in which LDOS interfaces the operating system with specific disk driver routines. This table is one of the examples of the versatility of the system. Ingenuity and oddball hardware will mix well to provide an easy interface. Pay particular attention to the fields indicating the allocation scheme for the drive. This data is an essential ingredient in the allocation and accessibility of file records.

The table contains a maximum of eight records - one for each logical drive designated 0-7. The TRS-80 Model III supports a standard configuration of four drives. This will be the default initialization when LDOS is BOOTed.

Here is the table layout:

DCT+0 :

The 1st byte of a 3-byte vector to the disk I/O driver routines. This would be an X'C3'. If the drive is disabled (see SYSTEM command), this will be an RET instruction (X'C9').

DCT+1 & DCT+2 :

This will contain the vector transfer address of the disk I/O routines driving the physical hardware.

DCT+3 :

Contains a series of flags for drive specifications. They are encoded as follows :

bit-7....Set to "1" if software write protected, "0" if not.

bit-6....Set to "1" for DDEN, set to "0" for SDEN

bit-5....Set to "1" if drive is 8" drive. If the drive is a 5-1/4" drive, the bit is "0".

bit-4....A "1" will cause the selection of the disk's second side. The first side will be selected if this bit is a "0". The bit value will match the side indicator bit in the sector header as written by the FDC.

bit-3....If this bit is set to a "1", it indicates a hard drive (Winchester). A "0" in this bit position denotes a floppy device (5-1/4" or 8").

bit-2....Used to indicate the time delay between selection of a 5-1/4" drive and the first poll of the status register. A "1" value indicates 0.5 seconds while a "0" value indicates 1.0 seconds. See the SYSTEM command for additional details.

If the drive is a hard drive, this bit will instead indicated either a fixed or removable disk, "0" = removable, "1" = fixed.

bits-1 & 0.These contain the step rate specification for the floppy disk controller. Again, see the SYSTEM command.

DCT+4 :

Contains additional drive specifications:

bit-7....This is reserved for future use. It should NOT be used, in order to maintain compatibility with future releases of LDOS.

bit-6....If "1", the controller is capable of double density mode.

Bit-5....A "1" denotes 2-sided operation while a "0" indicates single sided operation. Do not confuse this bit with Bit 4 of DCT+3. This bit shows that the disk is 2-sided. The other bit tells the controller what side the current I/O is to be on.

If the hard drive bit is set, a "1" denotes double the cylinder count stored in DCT+6.

bit-4....If "1", indicates an alien (non-standard) disk controller.

bits-0-through-3....This contains the physical drive address by bit selection (1, 2, 4, or 8). The system only supports a translation where more than one bit set is not permitted.

If the alien bit is set, these bits will indicate the starting head number.

DCT+5 :

Contains the current cylinder position of the drive. Its normal purpose is to store the track register of the FDC whenever the FDC is selected for access to this drive. It can then be used to reload the track register whenever the FDC is reselected.

If the alien bit is set, this byte will contain the drive select code for the alien controller.

DCT+6 :

This byte contains the highest numbered cylinder on the drive. Since cylinders are numbered from zero, a 35-track drive would be entered as X'22', a 40-track as X'27', and an 80-track as X'4F'.

If the hard drive bit is set, the true cylinder count will depend on DCT+4, bit 5. If that bit is a "1", this byte will be only half of the true cylinder count.

DCT+7 :

Contains certain allocation information:

bits-0-through-4.... Contain the highest numbered sector relative from zero. A ten-sector per track drive would show a X'09'. If DCT+4, bit 5 indicates 2-sided operation, the sectors per cylinder will be twice this number.

bits-5-through-7....contain the number of heads for a hard drive.

DCT+8 :

Contains additional allocation parameters:

bits-0-through-4....Contain the quantity of sectors per granule that was used in the formatting operation.

bits-5-through-7....Contain the quantity of granules per track allocated in the formatting process. If DCT+4, bit 5, indicates 2-sided operation, the granules per cylinder will be twice this number.

DCT+9 :

Contains the cylinder where the directory is located. For any directory access, the system will first attempt to use this value to read the directory prior to examining the BOOT sector directory storage byte in case the READ operation was unsuccessful.

It is essential that bytes DCT+6, DCT+7, and DCT+8 all relate without conflicts. That is to say, the highest numbered sector (+1) divided by the quantity of sectors per granule (+1), should equal the number of granules per track (+1).

D I R E C T O R Y R E C O R D S (DIREC)

The directory contains information required to access all files on the disk. The section containing directory records is limited to a maximum of 32 sectors due to physical limitations in the Hash Index Table. Two sectors are used for the Granule Allocation Table and the Hash Index Table. The directory is also contained completely on a single cylinder. Thus, a 10-sector per cylinder formatted disk will have, at most, 8 directory sectors. Consult the HIT documentation for the formula calculating the number of directory sectors.

A directory record is 32 bytes in length. Thus, each directory sector contains eight directory records. The first two directory records of the first eight directory sectors are reserved for system overlays. This is true even if the diskette does not contain an operating system (i.e. a data diskette). The total capacity of files is equal to the number of directory sectors times eight (since 256/32 = 8). The quantity available for use will always be reduced by 16 to account for those record slots reserved for the operating system. The following table shows the record capacity (file capacity) of each format type. The dash suffix on the density indicator represents the number of sides formatted:

	sectors/ cylinder	directory sectors	files per directory	avail for use
5" SDEN-1	10	8	64	48
5" SDEN-2	20	18	144	128
5" DDEN-1	18	16	128	112
5" DDEN-2	36	32	256	240
8" SDEN-1	16	14	112	96
8" SDEN-2	32	30	240	224
8" DDEN-1	30	28	224	208
8" DDEN-2	60	32	256	240
5" HARD	128	32	256	240
8" HARD	256	32	256	240

LDOS is upward compatible with other TRSDOS (tm) 2.3 compatible operating systems in its directory format. LDOS has further extended the data contained in the directory to add additional features and needed enhancements. The expert application programmer may find useful information in the directory - especially for those that write catalog programs. Since the directory information is so vital to the friendliness of programs, much information is displayed in the directory command as noted in other sections of this manual. A standard system vector has been included to either display an abbreviated directory or place its data in a user defined buffer area. For detailed information on this facility, see the @DODIR vector in the section on LDOS system vectors.

The following provides detailed information on the contents of each directory field:

DIR+0

This byte contains the entire attributes of the designated file. It is encoded as follows:

Bits 0-2 ... contain the access protection level of the file. The 3-bit binary value is encoded as follows:

0-FULL	1-KILL	2-RENAME	3-UNUSED
4-WRITE	5-READ	6-EXEC	7-NO ACCESS

Bit 3 ... Specifies the visibility; if "1", the file is INVisible to a directory display or other library function where visibility is a parameter. If a "0", then the file is declared VISible.

Bit 4 ... is used to indicate whether the directory record is in use or not. If set to "1", the record is in use. If set to a "0", the directory record is not active although it may appear to contain directory information. In contrast to other operating systems that zero out the directory record when you kill a file, LDOS only resets this bit to zero.

Bit 5 ... This bit is reserved for future use. Do not utilize it for any purpose if you want to maintain compatibility with future releases of LDOS.

Bit 6 ... A SYStem file is noted by setting this bit to a "1". If set to a "0", the file is declared a non-system file.

Bit 7 ... This flag is used to indicate whether the directory record is the file's primary directory entry (FPDE) or one of its extended directory entries (FXDE). Since a directory entry can contain information on up to four extents (see later notes on the extent fields), a file that is fractured into more than four extents requires additional directory records. If this bit is a "0", it implies it is an FPDE. If set to a "1", it is referencing an FXDE.

DIR+1

This contains various file flags and the month field of the packed date of last modification. It is encoded as follows:

Bit 7 ... This bit will be set to a "1" if the file was "CREATEd" (see CREATE Library Command). It will allocate a file that will never shrink in size. It will remain as large as its largest allocation. Since the CREATE command can reference a file that is currently existing but non-CREATEd, it can turn a non-CREATEd file into a CREATEd one. The same effect could be achieved by changing this bit to a "1".

Bit 6 ... If this flag is set to a "1", it indicates that the file has not been backed up since its last modification. The BACKUP utility is the only LDOS facility that will reset this flag. It is set during the close operation if the File Control Block (FCB+0, Bit 2) denotes a modification of file data.

Bit 5 ... This bit is reserved for future use. If you want to maintain compatibility with future releases of LDOS, do not utilize this bit for any purpose.

Bit 4 ... If the file was modified during a session where the system date was not maintained (remember that prompt during the BOOT?), this bit will be set to a "1" to indicate that the packed date of modification, if any, stored in the next fields is not the actual date when the modification occurred. If a "1", the directory command will display plus signs (+) between the date fields if the (A) option is requested.

Bits 0 through 3 ... contain the binary month of the last modification date. If this field is a zero, DATE was not set when the file was established nor since if it was updated.

DIR+2

This byte contains the remaining date of modification fields. They are encoded as follows:

Bits 3 through 7 ... contain the binary day of last modification.

Bits 0 through 2 ... contain the binary YEAR - 80. That is to say that 1980 would be coded as 000, 1981 as 001, 1982 as 010, and so forth.

DIR+3

Contains the end-of-file offset byte. This byte, and the ending record number (ERN), form a triad pointer to the byte position immediately following the last byte written. This also assumes that programmers, interfacing in machine language, properly maintain the next record number (NRN) offset pointer when the file is closed.

DIR+4

Contains the logical record length (LRL) specified when the file was initially generated or subsequently changed with a CLONE parameter.

DIR+5 through DIR+12

Contain the name field of the filespec. The file name will be left justified buffered with trailing blanks.

DIR+13 through DIR+15

Contain the extension field of the filespec. As in the name field, it is left justified buffered with trailing blanks.

DIR+16 & DIR+17

The UPDATE password hash code is contained in this field.

DIR+18 & DIR+19

The ACCESS password hash code is contained in this field. The protection level in DIR+0 is associated with this password.

DIR+20 & DIR+21

This field contains the ending record number (ERN) which is based on full sectors. If the ERN is zero, it indicates a file where no writing has taken place (or the file was not closed properly). If the LRL is not 256, the ERN value represents the sector where the EOF occurs. Actually, use ERN-l to account for a value relative to sector 0 of the file.

DIR+22 & DIR+23

This is the first extent field. Its contents tell you what cylinder stores the first granule of the extent, what relative granule it is, and how many contiguous grans are in use in the extent. It is encoded according to the following pattern:

DIR+22 Contains the cylinder value for the starting gran of that extent.

DIR+23, bits 0 through 4, contain the quantity of contiguous granules. The value is relative to 0. Therefore a "0" value implies one gran, "1" implies two, and so forth. Since the field is 5 bits, it contains a maximum of X'1F' or 31, which would represent 32 contiguous grans.

DIR+23, bits 5 through 7, contain the granule of the cylinder which is the first granule of the file for that extent. Again, this value is offset from zero.

DIR+24 & DIR+25

Contain the fields for the second extent. The format is identical to extent 1.

DIR+26 & DIR+27

Contain the fields for the third extent. The format is identical to extent 1.

DIR+28 & DIR+29

Contain the fields for the fourth extent. The format is identical to extent 1.

DIR+30

Is a flag noting whether or not a link exists to an extended directory record. If no further directory records are linked, the byte will contain X'FF'. If the value is X'FE', a link is recorded to an extended directory.

DIR+31

This is the link to the extended directory noted by the previous byte. The link code is the Directory Entry Code (DEC) of the extended directory record. The DEC is actually the position of the Hash Index Table byte mapped to the directory record. For additional information, see the section on the Hash Index Table.

EXTENDED DIRECTORY RECORDS

Extended Directory records (FXDE) have the same format as primary Directory records, except that only bytes 0 and 21 to 31 are utilized. Within byte 0, only bits 4 and 7 are significant. Byte 1 contains the DEC of the directory record which this is an extension of. An extended directory record may point to yet another directory record, so a file may contain an "unlimited" number of extents (limited only by the total number of directory records available, 256).

G R A N U L E A L L O C A T I O N T A B L E (GAT)

The Granule Allocation Table (GAT) contains information pertinent to the free and assigned space on the disk. The GAT also contains certain data specific to the formatting used on the diskette.

In order to deal with a wide range of hardware storage devices, an entire disk is partitioned into cylinders (tracks) and sectors: Each cylinder has a specified quantity of sectors. A group of sectors is allocated whenever additional space is needed. This group is termed a granule. The choice of a granule size is a compromise over minimum file lengths and overhead during the dynamic allocation process. The GAT is configured to provide for a maximum of eight granules per cylinder. In the allocation bytes, each bit set indicates a corresponding granule in use (or locked out). A reset bit indicates a granule free to be used.

In the GAT byte, bit 0 corresponds to the first relative granule. Bit 1 corresponds to the second relative granule. Bit 2 the third, and so on. A 5-1/4" single density diskette is formatted at 10 sectors per cylinder, 5 sectors per granule, 2 granules per cylinder. Thus, that configuration will use only bits 0 & 1 of the GAT byte. The remaining GAT byte will contain all l's - thereby denoting unavailable granules. Other formatting conventions are as follows:

	sectors/ cylinder	sectors/ granule	granules/ cylinder	maximum cylinders
5" SDEN	10	5	2	80
5" DDEN	18	6	3	80
8" SDEN	16	8	2	77
8" DDEN	30	10	3	77
5" HARD	128	16	8	153
8" HARD	256	32	8	256

This table assumes single sided media. LDOS supports double-sided operation within the confines of the hardware interfacing the physical drives to the CPU. A two-headed drive will function as a single unit with the second side as a cylinder for cylinder extension of the first side. A bit in the Drive Code Table (DCT) indicates one-sided or two-sided drive configuration.

A Winchester-type hard disk can be partitioned by heads into multiple logical drives. Information will be supplied along with the particular drive.

The Granule Allocation Table is the first relative sector of the directory cylinder. The following describes the layout of the GAT and the information contained in it.

GAT+X'00' through GAT+X'5F'

Contains the free/assigned table information. GAT+0 corresponds to cylinder 0, GAT+1 corresponds to cylinder 1, GAT+2 corresponds to cylinder 2, and so forth. As noted above, bit 0 of each byte corresponds to the first granule on the cylinder, bit 1 corresponds to the second granule, etc. A "1" indicates the granule is not available for use.

GAT+X'60' through GAT+X'BF'

Contains the available/locked out table information. It corresponds on a cylinder for cylinder basis as does the free/assigned table. It is used specifically during mirror-image backup functions to determine if the destination has the proper capacity to effect a backup of the source diskette.

GAT+X'C0' through GAT+X'CA'

Used in hard drive configurations by extending the free/assigned table from X'00' through X'CA'. Hard drives cannot be backed up in a mirror-image manner since their re-mapped cylinder configuration would exceed core limits. Thus, there is no need to reserve space for a lockout table. Hard drive capacity up to 202 mapped cylinders (404 standard) is supported.

GAT+X'CB'

Contains the operating system version used in formatting the disk. Disks formatted under LDOS 5.1 will have a value of X'51' contained in this byte. It is used to determine whether or not the diskette contains all of the parameters needed for LDOS 5.1 operation.

GAT+X'CC'

This byte contains the number of cylinders in excess of 35. Its use is to minimize the time required to compute the maximum cylinder formatted on the diskette. It was designed to be excess 35 so as to provide complete compatibility with alien systems not maintaining that byte. If you have a diskette that was formatted on an alien system for other than 35 cylinders, this byte can be automatically configured by using the REPAIR utility. See its reference in another section of this manual.

GAT+X'CD'

This byte contains data specific to the formatting of the diskette. Bit 6 set to "1" implies double density formatting. Bit 5 set to "1" indicates two-sided media. Bits 7, 4, and 3 are reserved for future assignment. Bits 2-0 contain the number of granules per cylinder -1.

GAT+X'CE' and GAT+X'CF'

Contains the 16-bit hash code of the disk master password. Its storage is in standard low-order high-order format.

GAT+X'D0' through GAT+X'D7'

Contains the diskette pack name. This is the name displayed at boot up if the diskette is a system diskette used for the boot operation. It is also the name displayed during a FREE or DIR. The name is assigned during the formatting operation or an ATTRIB disk renaming operation.

GAT+X'D8' through GAT+X'DF'

Contains the date that the diskette was formatted or the date that it was used as the destination in a backup operation. If the diskette is used during a BOOT, this date will be displayed adjacent to the pack name.

GAT+X'E0' through GAT+X'FF'

Contains the AUTO command buffer. This is the command that will be executed during a BOOT operation. If there is no AUTO command in place then GAT+X'E0' will contain an X'0D'.

H A S H I N D E X T A B L E (HIT)

The Hash Index Table is the key to addressing any file in the directory. It is designed so as to pinpoint the location of a file's directory with a minimum of disk accesses. A minimum quantity of disk accesses is useful to keep overhead low while providing rapid file access.

The procedure that the system uses to locate a file's directory is to first take the file name and extension and construct an 11-byte field with the file name left justified and padded with blanks. The file extension is then inserted, padded with blanks, and will occupy the three least significant bytes of the 11-byte field. This field is then processed through a hashing algorithm which produces a single byte value in the range X'01' through X'FF' (a hash value of X'00' is reserved to indicate a spare HIT position).

The hash code is then stored in the Hash Index Table (HIT) at a position corresponding to the directory record containing the file's directory. Since more than one 11-byte string can hash to identical codes, the opportunity for "collisions" exists. For this reason, the search algorithm will scan the HIT for a matching code entry, will then read the directory record corresponding to the matching HIT position, and will compare the filename/ext stored in the directory with that provided in the file specification. If both match, the directory has been found. If the two fields do not match, the HIT entry was a collision and the algorithm continues its search.

The position of the HIT entry in the hash table itself is called the Directory Entry Code (DEC) of the file. All files will have at least one DEC. Files that are extended beyond four extents will have DECs for each extended directory entry and use up more than one filename slot. Therefore, to maximize the quantity of file slots available, you should keep your files below five extents wherever possible.

Each HIT entry is mapped to the directory sectors by the DEC's position in the HIT. Conceptualize the HIT as eight rows of 32-byte fields. Each row will be mapped to one of the directory records in a directory sector. The first HIT row to the first directory record, the second HIT row to the second directory record, and so forth. Each column of the HIT field (the 0-31) is mapped to a directory sector. The first column is mapped to the first directory sector in the directory cylinder (not including the GAT and HIT). Therefore, the first column corresponds to sector number 2, the second column to sector number 3, and so forth. The maximum quantity of HIT columns actually used will be governed by the disk formatting according to the formula: N = number of sectors per cylinder minus two, up to a maximum of 32.

In the 5-1/4" single density configuration, there exist ten sectors per cylinder - two reserved for the GAT and HIT. Since only eight directory sectors are possible, only the first eight positions of each HIT row are used. Other formats will use more columns of the HIT, depending on the quantity of sectors per cylinder in the formatting scheme.

This arrangement works nicely when dealt with in assembly language for interfacing. Consider the DEC value of X'84'. If this value is loaded into the accumulator, a simple:

```
AND     1FH
ADD     A,2
```

will extract the sector number of the directory cylinder containing the file's directory. If that same value of X'84' was operated on by:

```
AND     0E0H
```

the resultant value will be the low-order starting byte of the directory record assuming the directory sector was read into a buffer starting at a page boundary. This procedure makes for easy access to the directory record.

Note that the first DEC found with a matching hash code may, in fact, be the file's extended directory entry (FXDE). It is therefore important, that if you are going to write system code to deal with this directory scheme, you properly deal with the FPDE/FXDE entries. See the section on directory records for additional information.

The following chart may help to visualize the correlation of the Hash Index Table to the directory records. Each byte value shown represents the position in the HIT. This position value is called the DEC. The actual contents of each byte will be either a X'00' indicating a spare slot, or the 1-byte hash code of the file occupying the corresponding directory record.

```
          ---------------- C O L U M N S ----------------
Row 1     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
          10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F

Row 2     20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
          30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F

Row 3     40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
          50 51 52 53 54 55 56 57 58 59 5A SB SC SD SE SF

Row 4     60 61 62 63 64 65 66 67 68 69 6A 68 6C 6D 6E 6F
          70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F

Row 5     80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
          90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F

Row 6     A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
          B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF

Row 7     C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
          D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF

Row 8     E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
          F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
          ---------------- C O L U M N S ----------------
```

The eight directory records for the directory cylinder, sector 2 would correspond to assignments in HIT positions 00, 20, 40, 60, 80, A0, C0, and E0. The following positions are reserved for system overlays:

00 -> BOOT/SYS	20 -> SYS6/SYS
01 -> DIR/SYS	21 -> SYS7/SYS
02 -> SYS0/SYS	22 -> SYSS/SYS
03 -> SYS1/SYS	23 -> SYS9/SYS
04 -> SYS2/SYS	24 -> SYS10/SYS
05 -> SYS3/SYS	25 -> SYS11/SYS
06 -> SYS4/SYS	26 -> SYS12/SYS
07 -> SYS5/SYS	27 -> SYS13/SYS

These entry positions, of course, correspond to the first two rows of each directory sector for the first eight directory sectors. Since the operating system accesses these overlays by position in the HIT rather than by file name, these positions are always reserved by the system.

The design of the Hash Index Table limits the system to a maximum support of 256 files on any one drive. With the current state of the art in disk drive technology, that limit is not considered to be of major impact – even considering large capacity Winchester drives supported by LDOS. This system will evolve to support the newer types of hardware coming to the market place, as they become available.

D I S K I / O T A B L E

LDOS interfaces with hardware peripherals by means of software drivers, The drivers are, in general, coupled to the operating system through data parameters stored in the system's many tables. In this manner, hardware not currently supported by LDOS may be easily supported by generating the appropriate driver software and updating the system tables.

Disk drive sub-systems, such as controllers for 5-1/4" drives, 8" drives, and hard disk drives, have many parameters addressed in the Drive Code Table (DCT). In addition to those operating parameters, controllers also require various commands to control the physical devices. These are commands such as SELECT, SECTOR READ, SECTOR WRITE, etc. LDOS has defined a standard linkage to deal with most commands available on standard Floppy Disk Controllers.

The resident system (SYS0) contains a series of entry points that deal with drivers linking to controllers. However, every function defined by LDOS is not contained in SYS0 since certain disk functions are not normally used in file access. This is not an undue restriction because it is not essential that all controller functions be routed through SYS0 routines. Certain controller function can be just as easily controlled from specialized application software.

The manner in which the driver controller linkage is established is by passing a function value contained in register "B" to the software driver that interfaces to the controller. Sixteen functions have been defined within LDOS. The following table briefly describes these functions:

HEX	DEC	FUNCTION	OPERATION PERFORMED
X'00'	0	NO Operation	Tests if drive is assigned in DCT
X'01'	1	SELECT	new drive and return status
X'02'	2	INIT	set to cylinder 0, restore, set side 0
X'03'	3	RESET	the Floppy Disk Controller
X'04'	4	RESTOR	issue FDC RESTORE command
X'05'	5	STEPIN	issue FDC STEP IN command
X'06'	6	SEEK	seek a cylinder
X'07'	7	TSTBSY	test if requested drive is busy
X'08'	8	RDHDR	read sector header information
X'09'	9	RDSEC	read sector command
X'0A'	10	VERSEC	verify if sector readable
X'0B'	11	RDCYL	issue an FDC cylinder read command
X'0C'	12	FORMAT	format the device
X'0D'	13	WRSEC	write a sector
X'0E'	14	WRSYS	write a system sector (e.g. directory)
X'0F'	15	WRCYL	issue an FDC cylinder write command

A detailed explanation of interfacing to various controllers is beyond the scope of this reference manual. Furthermore, the support staff cannot respond to questions relating to the design of specialized software drivers. It is to be understood that complex controller interfacing be undertaken only by those having the necessary assembly language skills.

S Y S T E M E N T R Y P O I N T S

****** I M P O R T A N T ******

The system entry points identified in this section are provided for use by skilled assembly language programmers. There is absolutely no attempt to provide an in-depth tutorial on how to write assembly language programs and routines using these entry points. Although more technical information is being provided in this reference manual than has been supplied with other operating systems, it does not replace other reference books covering the subject matter.

The information provided in this section will probably not be useful to the novice programmer. You will cause considerable trouble for yourself if you try to use it without FULLY understanding it. The experienced application programmer will find this an invaluable reference section. Since this information deals with specific functions of LDOS, every effort to verify the accuracy of the data has been undertaken. However, it is strongly suggested that you fully test your coding to be sure that the correct results are being produced prior to working with sensitive data.

This may very well be the first time that all of these system entry points have been identified by an "official source". You will also note the documentation of a number of routines located in the ROM of the TRS-80. Most of these have never been referenced by Radio Shack. We hope you find them informative.

A word of caution. Other operating systems designed to run on the TRS-80 may not support all of the system entry points identified herein. If you want to assure yourself that your application is compatible, you may have to contact the other suppliers of operating systems for their technical data (if they even provide it). However, you may want to make extensive use of this operating system so as to create sophisticated applications by utilizing the enhancements in LDOS.

Program Control Routines

@ABORT .... (Vector = X'4030')

This "JUMP" entry will cause an abnormal program exit and return to LDOS. Any JCL execution in progress will cease.

@EXIT .... (Vector = X'402D')

This is the normal "jump" vector to perform a program exit and return to LDOS.

LDOS Control Vectors

@ADTSK .... (Vector = X'403D')

This routine will add an interrupt level task to the real time clock task table. The task slot can be 0-11; however, most slots are already assigned to certain functions in LDOS. Slot assignments 0-7 are low priority tasks executing every 266.667 milliseconds, while slots 8-11 are high priority tasks executing every 33.334 milliseconds. See the STORAGE section for the slot assignments used by LDOS.

```
DE => Task Control Block (TCB)
 A => Task slot assignment
HL    Register pair is used
```

Note: The DE register is a pointer not to the location of your task driver, but to a block of RAM called the TCB, which contains the address of the task driver entry point. Upon entry to your task routine, the register IX will contain the TCB address.

@CMD .... (Vector = X'4296')

This "jump" vector will initiate a normal return to LDOS and accept a new command. It is identical to @EXIT.

@CMNDI .... (Vector = X'4299')

This "jump" vector performs an entry to the command interpreter. Your "command" line will be interpreted and executed just as if it was entered in response to an "LDOS Ready".

HL => points to the start of a line buffer containing your command string terminated with an <ENTER> (X'0D').

@DEBUG .... (Vector = X'440D')

This call vector will force the system to enter the DEBUGging package. A "G" command from the DEBUGger will continue program execution with the next instruction.

@ERROR .... (Vector = X'4409')

This vector will provide an entry to post an error message. Two options exist. @ERROR will normally terminate to the @ABORT function. If bit 7 of the accumulator is SET, the error message will be displayed and return will be made to the calling program. The second option will provide extended or abbreviated error messages. If bit 6 is not set, the complete error display is:

```
*** Errcod=xx, Error Message String ***
      <filespec or devicespec>
   Referenced at X'dddd'
```

whereas if bit 6 is set, then only the "Error message string" is displayed.

A => Error number with bits 6 & 7 optionally set.

@KLTSK .... (Vector = X'4046')

This routine, when called by an executing task driver, will remove the task assignment from the task table and return to the foreground application that was interrupted.

@RMTSK .... (Vector = X'4040')

This routine will remove an interrupt level task from the task control block table. It is entered by a CALL instruction and will return to the calling program.

A => task assignment slot (0-11) to remove

Register pairs HL and DE are used.

@RPTSK .... (Vector = X'4043')

This routine will exit the task process executing and replace the currently executing task vector address in the task control block table with the address following the CALL instruction. Return is made to the foreground application that was interrupted. This routine is entered with a CALL instruction.

DISK I/O PRIMITIVES

The system vectors contained in this section interface to the disk I/O driver. The address of the driver is contained in the Drive Code Table and can vary from drive to drive depending on your particular installation. Anyone having need to utilize these primitives must be thoroughly familiar with the function performed by each primitive. A discussion on this topic is beyond the scope of this manual. It is recommended that you obtain appropriate reference manuals relating to the particular controller in use for your disk system If your hardware is capable of double density operation, you most likely are using an FDC in the 179X series. These primitives are identified herein for such time as you have the knowledge and experience to utilize them.

SELECT .... (Vector = X'4754')

This vector will select a drive. LDRV$ and PDRV$ are updated. The appropriate time delay specified in your configuration (SYSTEM (DELAY=Y/N)) will be undertaken if the drive selection requires it.

C => logical drive number (0-7)
A <= error return code
Z <= set if no error

TSTBSY .... (Vector = X'4759')

This entry will perform a test of the last selected drive to see if it is in a busy state. If busy, it will be re-selected until it is no longer busy.

C => should contain the logical drive number

SEEK .... (Vector = X'475E')

This entry is used to seek a specified cylinder.

C => logical drive number
D => cylinder requested

WRSEC .... (Vector = X'4763')

This entry will write a sector to the disk

HL => buffer containing the sector of data
D => cylinder to write
E => sector to write
C => the logical drive number
A <= error return code
Z <= Set if no error

WRSYS .... (Vector = X'4768')

This entry will write a system sector (used in directory cylinder).

HL => buffer containing the sector of data
D => cylinder to write
E => sector to write
C => the logical drive number
A <= error return code
Z <= set if no error

WRCYL .... (Vector - X'476D')

This entry is used to write an entire cylinder of properly formatted data. The data format must conform to that identified in your controller's reference manual.

HL => pointer to format data
D => cylinder to write
C => the logical drive number
A <= error return code
Z <= set if no error

VERSEC .... (Vector = X'4772')

This entry will verify a sector without transferring any data from disk to the buffer.

D => cylinder to verify
E => sector to verify
C => the logical drive number
A <= error return code
Z <= set if no error

RDSEC .... (Vector = X'4777')

This entry will transfer a sector of data from the disk to your buffer.

HL => pointer to the buffer to receive the sector
D => cylinder to read
E => sector to read
C => the logical drive number
A <= error return code
Z <= set if no error

File Control Routines

@CLOSE ... (Vector = X'4428')

This routine will close a file or device. When a file is closed, the directory is updated which is essential. All files that have been written to must be closed.

```
DE => File or Device Control Block
 A <= Error return code
 Z <= Set if no error existed
```

@FEXT .... (Vector = X'444B')

This routine will insert a default file extension into the FCB if the file specification entered contains no extension. This must be done before the file is opened.

```
DE => File Control Block
HL => Pointer to default extension (3 characters)
```

@FSPEC .... (Vector = X'441C')

This routine will fetch a file or device specification from an input buffer. Conversion of lower case to upper case will be made.

```
HL => buffer containing file specification
DE => 32-byte File Control Block
HL <= points to the terminating character found
DE <= points to the beginning of FCB
 A <= the terminating character
 Z <= set if valid file specification found
```

@INIT .... (Vector = X'4420')

INIT will open an existing file. If the file is not found, it will be created according to the file specification.

```
HL => 256-byte disk I/O buffer
DE => File Control Block containing the file specification
 B => Logical Record Length to be used while the file is open.
HL <= Remains unchanged
DE <= Remains unchanged
 A <= Error return code
CF <= Set if a new file was created
 Z <= Set if no error detected
```

@KILL .... (Vector = X'442C')

This routine will kill a file or device. If a file is to be killed, the FCB must be in an open condition. The file's directory will be updated and the space occupied by the file will be de-allocated. The file name, extension, and drive number will be placed back into the FCB. If a device, it will be removed from the device control block tables if it is not a system device. The device should first be RESET.

```
DE => File Control Block
 A <= error code returned
 Z <= set if no error detected
```

@OPEN .... (Vector = X'4424')

This routine will open an existing file or device.

```
HL => Buffer for disk I/O (256 bytes)
DE => File control block containing the filespec
 B => Logical record length for the open file
 A <= Error return code
 Z <= Set if open was successful
```

Program Loading Control Routines

@LOAD .... (Vector = X'4430')

This routine will load a program file (a file in load module format).

```
DE => FCB containing the filespec of the file to load.
 A <= error return code
 Z <= set if load was successful
HL <= entry address (TRA) retrieved from file
```

@RUN .... (Vector = X'4433')

This routine will load and execute a program file. If any errors are encountered during the load, the system will jump to @ERROR, print the appropriate message, and terminate by jumping to @ABORT. If @RUN is entered by a CALL instruction, the stack will contain the return address upon entry to the program. If this is not needed, a JP instruction may be used to enter @RUN.

```
DE => FCB containing the filespec of the file to RUN.
```

Disk file handler routines
==========================

In general, only the flag register and the accumulator are disturbed unless a register is used to return data requested by the system call. All file access routines are accessed by a CALL instruction.

```
@BKSP .... (Vector = X'4445')
------------------------------
   This routine will perform a backspace of one logical record.

   DE => FCB of the file to backspace
    A <= Error return code
    Z <= Set if the operation was successful

@CKEOF .... (Vector = X'4458')
------------------------------
   Check for the end-of-file at the current logical record number.

      DE => FCB for the file to check
       A <= Error return code
       Z <= Set if not at the end of file

@LOC .... (Vector = X'446D')
----------------------------
   Calculate the current logical record number.

   DE => FCB for the file to check
   BC <= the logical record number
    A <= Error return code
    Z <= Set if the operation was successful

@LOF .... (Vector = X'4470')
----------------------------
   Calculate the EOF logical record number.

      DE => FCB for the file to check
      BC <= the logical record number
       A <= Error return code
       Z <= Set if the operation was successful

@PEOF .... (Vector = X'4448')
-----------------------------
   This routine will position an open file to the end-of-file position. An
   end-of-file-encountered error (X'1C') will be returned if the file is
   already position to the end. Your program may ignore this error.

      DE => FCB of the file to position
       A <= Error return code
       Z <= Set if the operation was successful
```

@POSN .... (Vector = X'4442')

Position a file to a logical record. This will be useful for positioning to records of a random access file. When the @POSN routine is used, Bit 6 of FCB+1 is automatically set to ensure that the EOF will be updated when the file is closed only if the NRN exceeds the current ERN.

```
DE => FCB for the file to position
BC => the logical record number
 A <= Error return code
 Z <= Set if the operation was successful
```

@READ .... (Vector = X'4436')

Read a logical record from a file. If the LRL defined at open time was 256 (0), then the NRN sector will be transferred to the buffer established at open time. For LRL between 1 and 255, the next logical record will be placed into the user record buffer, UREC. The 3-byte NRN is updated after the read operation.

```
DE => FCB for the file to read
HL => UREC (needed if LRL <> 0) - (unused if LRL=256)
 A <= Error return code
 Z <= Set if the operation was successful
```

@REW .... (Vector = X'443F')

This routine will rewind a file to its beginning and reset the 3-byte NRN to 0. The next record to I/O will be the first record of the file.

```
DE => FCB for the file to rewind
 A <= Error return code
 Z <= Set if the operation was successful
```

@RREAD .... (Vector = X'445E')

This routine will force a reread of the current sector to occur before the next i/o request is serviced. Its most probable use would be in applications that reuse the disk I/O buffer for multiple files to make sure that the buffer contains the proper file sector. This routine is only valid for byte I/O or blocked files.

```
DE => FCB for the file to reread
 A <= Error return code
 Z <= Set if the operation was successful
```

@RWRIT .... (Vector = X'4461')

This routine will rewrite the current sector following a write operation. The @WRITE function advances NRN after the sector is written. @REWRIT will decrement the NRN and write the disk buffer again.

```
      DE => FCB for the file to rewrite
       A <= Error return code
       Z <= Set if the operation was successful

@SKIP .... (Vector = X'4464')
-----------------------------
   This routine  will  cause a skip past the next logical record. Only  the
   record number contained  in  the  FCB will  be changed. No physical  I/O
   will take place.

      DE => FCB for the file to skip
       A <= Error return code
       Z <= Set if the operation was successful

@VER .... (Vector = X'443C')
----------------------------
   This routine performs a @WRITE operation followed by a test  read of the
   sector  (assuming  that the WRITE required physical  I/O) to verify that
   it will be readable.

      DE => FCB for the file to verify
      HL <= user buffer containing the logical record
       A <= Error return code
       Z <= Set if the operation was successful

@WEOF .... (Vector = X'445B')
-----------------------------
   This routine will force  the  system to update the directory  entry with
   the current end-of-file information.

      DE => FCB for the file to WEOF
       A <= Error return code
       Z <= Set if the operation was successful

@WRITE .... (Vector = X'4439')
------------------------------
   This routine will cause a  write to  the next  record identified in  the
   FCB.  If  LRL is  less than 256, then the  logical  record  in the  user
   buffer will be transferred to the file. If LRL is  equal to  256, a full
   sector write will  be made using the disk  I/O buffer identified at file
   open time.

      HL <= UREC - the user record buffer - (unused if LRL=256)
      DE => FCB for the file to write
       A <= Error return code
       Z <= Set if the operation was successful
```

General Purpose Routines

@CKDRV .... (Vector = X'4290')

This routine will check a drive reference to ensure that the drive is in the system and a formatted diskette is in place.

```
    C => Logical drive number
    A <= Error return code.
    Z <= If drive is ready.
   NZ <= If drive is not ready
   CF <= Set if disk is write protected.
```

@DATE .... (Vector = X'3033')

Get today's date in display format (xx/xx/xx)

```
   HL => Buffer area to receive date string.
```

@DODIR .... (Vector = X'4419')

This routine will read visible files from a disk directory, or find the free space on a disk. The display to the screen will be in a 4 across format.

```
    C => Logical drive number. (THIS REGISTER USED IN ALL EXAMPLES).


    B => If 0, will display the directory to *DO, if 1 will send the
         directory to a buffer.
   HL => Buffer to receive directory.

    B => If 2, will display the directory to *DO. If 3, will send the
         directory to a buffer.
   HL => 4 character extension /$$$. The / is mandatory. Any of the
         extension characters not specified must be replaced with a $.
         If B = 3, then HL also points to the buffer.

    B => If 4, will get free space on the disk.
   HL => 20 character buffer. The information will be:

         bytes  1-16 = disk name and date
         bytes 17-18 = Total K originally available
         bytes 19-20 = Free K available
```

@DSPLY .... (Vector = X'4467')

This routine will display a message line. The line must be terminated with either an (ENTER> (X'0D') or an ETX (X'03'). If an ETX terminates the line, the cursor will be positioned immediately after the last character displayed.

HL => points to the 1st byte of your message.
HL is returned unchanged.

@FNAME .... (Vector = X'4293')

Recover the file name & extension from the directory.

DE => Buffer to receive file name/ext
B => DEC of file desired
C => drive number of drive containing the file

@LOGER .... (Vector = X'428D')

Issue a log message to the Job Log. Message is any character string terminating with an <ENTER> (X'0D').

HL => points to the first character of the message line.
HL is returned unchanged.

@LOGOT .... (Vector = X'428A')

Display and log a message. This will perform the same function as @DSPLY followed by @LOGER.

HL => points to the first character of the message line.
HL is returned unchanged.

@MSG .... (Vector = X'4402')

Message line handler to send a message to any device.

DE => Device or File Control Block to receive output
HL => pointer to the message line
HL is returned unchanged.

@PARAM .... (Vector = X'4454')

This routine can be used to parse an optional parameter string. Its primary function is to parse command parameters contained in a command line totally enclosed within parentheses. Acceptable parameter format is:

```
PARM=X'dddd' .... hexadecimal entry
PARM=ddddd   .... decimal entry
PARM="string" ... alphanumeric entry
PARM= flag   .... ON, OFF, Y, N, YES or NO
```

The 2-byte memory area pointed to by the address field of your table receives the value of PARM if PARM is non-string. If a string is entered, the 2-byte memory area receives the address of the 1st byte of "string". The entries ON, YES and Y return a value of X'FFFF' while OFF, NO and N will return a X'0000'. if a parameter name is specified on the command line followed by an equal sign and no value then a X'0000' or NO will be returned.

If a parameter name is used on the command line without the equal then a value of X'FFFF' or ON will be assumed. For any allowed parameter that is completely omitted on the command line, the 2 byte area will remain unchanged.

The valid parameters are contained in a user table which must be of the following format:

A 6-character "word" left justified and buffered by blanks followed by a 2-byte address vector to receive the parsed values. Word and vector may be repeated for as many parameters as are necessary. A byte of X'00' must be placed at the end of the table to indicate its ending point.

DE => beginning of your parameter table.
HL => command line to parse
Z <= Set if either no parameters found or valid parameters.
NZ <= If a bad parameter was found

@PRINT .... (Vector = X'446A')

This routine will output a message line to the printer. The message must conform to the syntax specified under @DSPLY.

HL => Pointer to the message to be output
HL is returned unchanged

@TIME .... (Vector = X'3036')

Get the time of day in display format (XX:XX:XX)

HL => Buffer to receive the time string.

ROM Resident I/O routines and vectors

@CTL .... (Vector = X'0023')

This routine will output a control byte to a logical device or a file. If a device control block is referenced, the TYPE byte must permit CTL operation.

```
DE => DCB or FCB to control output
 A => Byte to output
```

@DSP .... (Vector = X'0033')

This routine will output a byte to the video display.

```
 A => Byte to display
```

@GET .... (Vector = X'0013')

This routine will fetch a byte from a logical device or a file.

```
DE => FCB
 A <= Byte fetched or error return code if disk error
 Z <= set if byte was fetched from disk without error

DE => DCB
 A <= Byte fetched or 0 if no byte available.
 Z <= Set if no byte available.
```

@KBD .... (Vector = X'002B')

Scan the keyboard and return the keyboard character, if a key is pressed. If no key was pressed, a zero value will be returned.

```
 A <= Contains the value of the key depressed
CF <= will be set if the control key (SHIFT-DOWN ARROW) was pressed.
MF <= high bit of returned byte will be set if (CLEAR) is pressed.
```

@KEY .... (Vector = X'0049')

This routine will scan the keyboard and return with the key depressed. It will not return until a key is depressed.

```
 A <= the character entered
```

@KEYIN .... (Vector = X'0040')

This routine will accept a line of input until terminated by either an <ENTER> or <BREAK>. During the input, the routine will display the entries. Backspace, tab, line delete, and 32 cpl mode are supported.

```
HL => user line buffer of length = B+1
 B => maximum number of characters to input
HL <= points to buffer start
 B <= the actual number of characters input
CF <= set if <BREAK> terminated the input
```

@PAUSE .... (Vector = X'0060')

This routine will suspend program execution and go into a "wait" state. The delay is approximately 14.796 microseconds per count.

```
BC => delay count
 A register is used.
```

@PRT .... (Vector = X'003B')

This routine will output a byte to the printer. If a zero value is passed, then the printer status will be returned. This method is recommended over checking the port directly.

```
 A => the character to print
 A <= the printer status if a zero was output
```

@PUT .... (Vector = X'001B')

This routine will output a byte to a logical device or a file.

```
DE => Device or File Control Block of the output device
 A => the byte to output
 A <= error return code if disk output
 Z <= if output to disk without error
```

@WHERE .... (Vector = X'000B')

Vector used to resolve relocation address of calling routine.

```
HL <= the address following the CALL instruction
```

Special Purpose Routines - Miscellaneous

GETDCT .... (Vector = X'478F')

This routine will obtain the address of the drive code table for the requested drive.

```
    C => logical drive number (0-7)
   IY <= the DCT address
```

DCTBYT .... (Vector = X'479C')

This entry will recover a byte field from the drive code table.

```
    C => logical drive number (0-7)
    A => relative byte desired in the table (3-9)
    A <= content of the DCT field requested
```

DIRRD .... (Vector = X'4B10')

This entry will read a directory sector containing the directory entry for a specified Directory Entry Code (DEC). The sector will be written to the system buffer, SBUF$, and the register pair HL will point to the first byte of the directory entry specified by the DEC.

```
    B => Directory Entry Code of the file
    C => Logical drive number (0-7)
   HL <= points to the DEC's directory entry
    A <= error return code
    Z <= set if no error
```

DIRWR .... (Vector = X'4B1F')

This entry will write the system buffer, SBUF$, back to the disk directory sector that contains the directory entry of the DEC specified in the calling linkage.

```
    B => Directory Entry Code of the file
    C => Logical drive number (0-7)
    A <= error return code
    Z <= set if no error
```

RDSSEC .... (Vector = X'4B45')

This entry will read the system sector identified by the calling linkage.

```
   HL => pointer to the buffer to receive the sector
    D => cylinder to read
    E => sector to read
    C => the logical drive number
    A <= error return code
    Z <= set if no error
```

DIRCYL .... (Vector = X'4B64')

This entry will recover the cylinder number containing the directory for the requested drive. It is identical to calling the DCTBYT entry with an argument of 9 (A=9).

```
    D <= returns the cylinder number of the directory
```

MULTEA .... (Vector = X'4B6B')

This entry will perform an 8-bit by 8-bit unsigned integer multiplication. It is assumed that the result will not overflow a 8-bit field since the routine is only used on small integer values.

```
    A => multiplicand value
    E => multiplier value
    A <= resultant value
```

Note: Register D is used

DIVEA .... (Vector = X'4B7A')

This routine performs an 8-bit unsigned integer divide.

```
    E => dividend value
    A => divisor value
    A <= resultant value
    E <= remainder
```

MULT .... (Vector = X'444E')

This routine will perform an unsigned integer multiplication of a 16-bit multiplicand by an 8-bit multiplier. The resultant value is stored in a 3-byte register field.

```
   HL => multiplicand value
    A => multiplier value
   HL <= two high order bytes of resultant value
    A <= low-order byte of the resultant value
```

Note: Register pair DE is used

DIVIDE .... (Vector = X'4451')

This routine will perform a division of a 16-bit unsigned integer by a 8-bit unsigned integer.

```
   HL => dividend value
    A => divisor value
   HL <= resultant value
    A <= remainder value
```

E R R O R D I C T I O N A R Y

The Operating System Error Dictionary is contained in the SYS4 system overlay. System errors will normally result in the display of messages contained in the following list. Information on the accessibility of the dictionary messages to the assembly language programmer is contained in the section on system vector entry points.

Error	Displayed Message
0	No error

This indicates that the @ERROR routine was called without any error condition being detected. A return code of zero indicates no error.

1 Parity error during header read X'01'

During a sector I/O request, the system was unable to satisfactorily read the sector header. Repeated failures would most likely indicate media or hardware failure.

2 Seek error during read X'02'

During a read sector disk I/O request, the requested cylinder that should contain the sector was not located within the time period allotted by the step rate specified in the drive code table. Either the cylinder is not formatted on the diskette, or the step rate designated is too low a value for the hardware to respond.

3 Lost data during read X'03'

During a read sector request, the CPU was late in accepting a byte from the FDC data register. For more information, consult the reference manual for the floppy disk controller used in your disk controller.

4 Parity error during read X'04'

During a read sector disk I/O request, the FDC sensed a CRC error. Possible media failure would be indicated. The Drive hardware could also be at fault.

5 Data record not found during read X'05'

A disk sector read request was generated with a sector number not found on the cylinder referenced.

6 Attempted to read system data record X'06'

A read request for a sector located within the directory cylinder was made without using the directory read routines. Directory cylinder sectors are written with a data address mark that differs from the data sector's data address mark. Consult the controller reference manuals for information concerning address marks.

7 Attempted to read locked/deleted data record X'07'

In systems that support a "deleted record" data address mark, this error message will appear if a "deleted" sector is requested to be read. LDOS currently does not utilize the "deleted" sector data address mark.

8 Device not available X'08'

A reference was made for a logical device that could not be located in the device control blocks. Most likely your device specification was in error. A DEVICE command can be used to display all devices available to the system.

9 Parity error during header write X'09'

This is the same type of error as error 1 except that the operation requested was sector WRITE.

10 Seek error during write X'0A'

This is the same type of error as error 2 except that the operation requested was sector WRITE.

11 Lost data during write X'0B'

During a sector write request, the CPU was not fast enough in transferring a byte to the FDC so it could be written to the disk.

12 Parity error during write X'0C'

A CRC error was generated by the FDC during a sector write operation.

13 Data record not found during write X'0D'

Similar to error 5, the sector number requested for the write operation could not be located on the cylinder being referenced. Either the request is erroneous, or the cylinder is improperly formatted.

14 Write fault on disk drive X'0E'

This error message results when the FDC returns a "write fault" error. Consult your FDC reference manual.

15 Write protected disk X'0F'

A write request was generated to a disk which had a write protected diskette, or was software write protected. On 5-1/4" diskettes, a covered up notch will protect the diskette from being written. On 8" media, an exposed notch will perform the same thing. If you want to write on a diskette, you must observe the proper notch condition.

16 Illegal logical file number X'10'

A bad Directory Entry Code (DEC) was found in the File Control Block (FCB). This usually indicates that your program has altered the FCB improperly.

17 Directory read error X'11'

Any disk error sensed during the reading of directory information will result in this error. It could be media failure, hardware failure, or program crashes.

18 Directory write error X'12'

Similar to error 17 but the error condition is sensed while attempting to write a directory sector back to the disk. The integrity of the directory is now suspect. This error can also occur if a disk is write protected when attempting a directory write.

19 Illegal file name X'13'

The file specification provided to the system has a character not conforming to the file specification syntax. See the reference manual section on filespecs.

20 GAT read error X'14'

Any disk error sensed during the reading of the granule allocation table will result in this error. It could be media failure, hardware failure, or program crashes.

21 GAT write error X'15'

Similar to the above except that the error was sensed during a WRITE request. The integrity of the GAT is suspect. This error may also occur if the disk was write protected during a GAT write attempt.

22 HIT read error X'16'

Similar to error 20 but occurring during the reading of the Hash Index Table.

23 HIT write error X'17'

Similar to error 21 but occurring during the writing of the Hash Index Table.

24 File not in directory X'18'

You referenced a file specification that could not be located in the directory. Note that if your request was to LOAD or RUN the file, the error message displayed would be "Program not found". Most likely the cause was a misspelled filespec.

25 File access denied X'19'

The requested file was password protected and the password used was neither the ACCESS nor UPDATE password. This will also occur if a password is specified for a non-password protected file.

26 Full or write protected disk X'1A'

An open of a new file was requested and the target disk's directory was entirely in use. Use another diskette or kill off unneeded files. The error could also result if the disk was protected from write requests.

27 Disk space full X'1B'

While a file was being written, all available space on the disk was allocated before the file was completely written. Whatever space was already allocated to the file will still be allocated although the file's end of file pointer will not be updated. It may be desirable to kill the file to recover the space after writing the file to another diskette.

28 End of file encountered X'1C'

You attempted to read past the end of file pointer. The file was smaller than your application thought. This error can also be used within an application to determine the end of a sequentially read file.

29 Record number out of range X'1D'

A request to read a record within a random access file (see the @POSN vector) provided a record number that was beyond the end of the file.

30 Directory full - can't extend file X'1E'

This will result whenever a file has in use all extent fields of its last directory record, and must find a spare directory slot but none is available. All available file positions are in use. See the technical section on directory records for more information. The solution would be to repack the disk by individually copying its files to a freshly formatted diskette.

31 Program not found X'1F'

The load request for a file can not be completed because the file was not located in the directory. Either the filespec was misspelled or the disk that contained the file was not mounted.

32 Illegal drive number X'20'

This error will occur whenever a reference is made to a disk drive that is not included in your system (see the DEVICE command) or the drive requested was not ready for access (no diskette, drive door open, etc.).

33 No device space available X'21'

The SET command was used to establish a new device in the system. Unfortunately, all of the resident system area reserved for Device Code Block tables is already in use. It is suggested that you use the DEVICE command to see if any currently defined non-system devices can be eliminated.

34 Load file format error X'22'

An attempt was made to LOAD a file that did not conform to the format structure for a program file capable of being loaded by the system loader. Most likely, the file referenced is a data file or a BASIC program file.

35 Memory fault X'23'

During the process of loading a program file, the integrity of the load is monitored to ensure that each memory position loaded stores the proper byte value. In cases of partial memory failure, one or more bits of a memory cell will not replicate the value being loaded. This error will then be displayed. If this condition repeats, it is suggested that you subject your hardware to a thorough memory test to locate the root cause.

36 Attempted to load read only memory   X'24'

The program file being loaded referenced a memory cell that could not be altered. Either the cell was part of the read only memory (ROM), or the address was referencing an area of the machine not containing any read/write memory (RAM). Use CMDFILE to locate the address loading information of the program file and verify the memory available in your CPU.

37 Illegal access attempted to protected file   X'25'

The ACCESS password was given for an operation that required the UPDATE password.

38 File not open   X'26'

An I/O operation was requested using a File Control Block that indicated a closed file. See the section on File Control Blocks and the field FCB+0.

39 Device in use   X'27'

A request was made to KILL a device (remove it from the Device Control Block tables) while it was in use. It is necessary to first RESET a device in use.

40 Protected system device   X'28'

You cannot kill any of the following devices: *KI, *DO, *PR, *JL. If you try, you will get this error message.

-- Unknown error code

Any time the error routine is called with a error number not in the acceptable range, this message will be displayed. It most likely indicates a software problem.

F I L E C O N T R O L B L O C K (FCB)

The File Control Block (FCB) is a 32-byte region that is used by the system to interface with a file that has been "opened". Its contents are extremely dynamic. As records are written to or read from the disk file, specific fields in the FCB are modified. It is extremely important that during the time period that a file is open, you avoid changing the contents of the FCB unless you are sure that its alteration will in no way effect the integrity of the file.

During most system access of the FCB, the IX index register is used to reference each field of data. Register pair DE is used primarily for the initial reference to the FCB address. The information contained in each field of the FCB follows:

FCB+0

Contains the TYPE code of the control block.

Bit 7 ... If set to "1", will indicate that the file is in an open condition; if set to "0", the file is assumed closed. This bit can be tested to determine the "open" or "closed" status of a FCB.

Bits 6-3 ... reserved for future use.

Bit 2 ... will be set to "1" if any WRITE operation was performed by the system on this file. It is used specifically to update the MOD flag in the directory record when the file is closed.

Bits 1 and 0 ... reserved for future use.

FCB+1

Contains status flag bits used in read/write operations by the system.

Bit 7 ... If set to a "1", it indicates that I/O operations will be either full sector operations or byte operations of logical record length (LRL) less than 256. If set to a "0", only sector operations will be performed. If you are going to utilize only full sector I/O, system overhead is reduced by specifying the LRL at open time to be 0 (indicating 256). An LRL of other than 256 will set bit 7 to a "1" on open.

Bit 6 ... If set to a "1", it indicates that the end-of-file (EOF) is to be set to ending-record-number (NRN) only if NRN exceeds the current value of EOF. This is the case if random access is to be utilized. During random access, the EOF will not be disturbed unless you are extending the file beyond the last record slot. Any time the position vector (@POSN) is called, it automatically sets bit 6. If bit 6 is set to a "0", then EOF will be updated on every WRITE operation.

Bit 5 ... If set to a "0", then the disk I/O buffer contains the current sector denoted by NRN. If set to a "1", then the buffer does not contain the current sector. During byte I/O, bit 5 is set when the last byte of the sector is read. A sector read will reset the bit & show the buffer to be current.

Bit 4 ... If set to a "1", it indicates that the buffer contents have been changed since the buffer was read from the file. It is used by the system for determining whether the buffer must be written back to the file before reading another record. If set to a "0", the indication is that no buffer modification was performed.

Bit 3 ... Is used to specify that the directory record is to be updated every time that the NRN exceeds the EOF. Normal operation is to update the directory only when a FCB is closed. Some unattended operations may utilize this extra measure of file protection. It is specified by appending an exclamation mark "!" to the end of a filespec when the filespec is requested at open time.

Bits 2-0 ... Contain the access protection level as retrieved from the directory of the file. For specific bit patterns, see the directory record explanation.

FCB+2

Is reserved by the system for future use.

FCB+3/4

Contain the buffer address in lo-order - hi-order format. This is the buffer address specified in register pair HL at open time.

FCB+5

Contains the relative byte offset within the current buffer for the next I/O operation. If this offset is a zero value) then FCB+1, Bit 5 must be examined to determine if the 1st byte in the current buffer is the target position or the 1st byte of the next record. If you are performing sector I/O of byte data (i.e. maintaining your own buffering)) then it is important to maintain this byte when you close the file if the true end-of-file is not at a sector boundary.

FCB+6

Contains the logical drive number in binary of the drive containing the file. It is absolutely essential that this byte be left undisturbed. It, and FCB+7 are the only links to the directory information for the file.

FCB+7

Contains the directory entry code (DEC) for the file. This code is the relative position in the hash index table where the hash code for the file appears. Do not tamper with this byte. It, and FCB+6, are needed to properly close the file.

FCB+8

Contains the end-of-file byte offset. This byte is similar to FCB+5 except it pertains to the end-of-file rather than the next-record-number.

FCB+9

Contains the logical record length in effect when the file was opened. This may not be the same LRL that exists in the directory. The directory LRL is generated at the file creation and will never change unless another file is cloned to it.

FCB+10/11

Contain the next-record-number (NRN), which is a pointer to the next I/O operation. When a file is opened, NRN is zero indicating a pointer to the beginning. Each sequential sector I/O advances NRN by one.

FCB+12/13

ERN of the file. This is a pointer to the sector that contains the end-of-file indicator. In a null file (one with no records), ERN will be equal to 0. If one sector had been written, ERN would be equal to 1.

FCB+14/15

Contains the same information as the first extent of the directory. This represents the starting cylinder of the file (FCB+14) and the starting relative granule within the starting cylinder (FCB+15). FCB+15 also contains the number of contiguous granules allocated in the extent. This will always be used as a pointer to the beginning of the file referenced by the FCB.

FCB+16-FCB+19

This is a 4-byte quad that contains cumulative granule allocation information for an extent of the file. Relative bytes 0 & 1 contain the cumulative number of granules allocated to the file up to but not including the extent referenced by this field. Relative byte 2 contains the starting cylinder of this extent. Relative byte 3 contains the starting relative granule for the extent and the number of contiguous granules.

FCB+20-FCB+23

Contain information similar to the above but for another extent of the file.

FCB+24-FCB+27

Contain information similar to the above but for a third extent of the file.

FCB+28-FCB+31

Contain information similar to the above but for a fourth extent of the file.

The file control block contains information only on four extents at any one time. If the file has more than four extents, additional directory accessing will be done to shift the 4-byte quads to make space for the new extent information. Although the system can handle a file of any number of extents, it is wise to keep the total number of extents small. The most efficient file is one with a single extent. The number of extents can be reduced by copying the file to a diskette containing a great deal of free space.

F I L E F O R M A T S

Disk Command File Format

The disk command file (load module) format consists of the following structure:

a. Byte X'05' indicates the FILENAME field, followed by a byte indicating the FILENAME length (typically a six-byte field).

b. Byte X'01' indicates a start-of-block, followed by a one-byte length of block, where the count includes the two byte load address. Following this is the two-byte block load address, followed by the block itself. A block length of X'00' indicates a 254 byte block plus two bytes for the load address. A block length of X'01' indicates a 255 byte block plus two bytes for the load address (0FFH + 02H = 01H). Similarly, a block length of X'02' indicates a 256 byte block plus two bytes for the load address (00H + 02H = 02H). Thus, the actual code block length can always be obtained by subtracting two from the value specified. This is repeated for as many blocks as are in the file.

c. Byte X'07' indicates that the following will be a PATCH NAME field. It is followed by the length byte for the actual field containing the PATCH NAME (in ASCII). This PATCH HEADER will be followed by the actual load blocks of the PATCH. When a patch is YANKed or removed the load blocks following the PATCH HEADER will begin with a X'10', which will cause the loader to ignore the blocks.

d. An X'02' is written to indicate the end of the code and the beginning of the transfer address. It is followed by an X'02' block length, then the two-byte transfer address.

e. The standard byte used to indicate a comment is X'1F'. A comment may appear between any two blocks in a file.

NOTE: Those header bytes that have not been defined above are reserved for system use. Some are used in LDOS 5.1, while others are planned for future use. Any byte larger than X'1F' encountered where a header is expected by the system loader will cause a "Load file format error" message.

Tape File Object Code Format

A SYSTEM tape has a similar format; however, the control bytes are different. The SYSTEM file structure is as follows:

a. X'55' indicates the start of the FILENAME, followed by the six character file name (buffered with blanks, X'20' to fill out to six characters).

b. X'3C' indicates the start of a block, followed by a one-byte block length. This, in turn, is followed by the two-byte block load address. The tape block length does not include the load address. The block of code follows. Immediately following the code block is a one-byte checksum determined by the modulo 256 sum of each byte in the block plus each byte of the load address (modulo 256 is achieved by performing 8-bit register addition ignoring all carries out of the high-order bit).

c. (b) is repeated for as many blocks as are needed An X'78' is written to indicate the end-of-file, followed by the two-byte transfer address.

F I L T E R S and D R I V E R S

LDOS contains command level procedures that provide easy access to device references so that modifications may be made to the way in which devices are treated by the system. All devices require some type of driving program (routine) that is used to handshake the device with the system and cater to the special features and requirements of the device hardware. Some drivers are already implemented within either the ROM interpreter or the operating system to handle standard devices. For instance, the ROM includes drivers for handshaking the keyboard, video, and parallel printer. LDOS contains drivers for operating the RS-232 port.

Some devices are completely supported with the existing drivers in the total LDOS environment. Other devices may need a little more support. The characteristics of a driver may be modified by introduction of a FILTER. For instance, suppose your printer required a line feed upon receipt of a carriage return to advance the paper. The ROM printer driver does not provide this function. Instead of writing a completely new printer driver, only a filter need be included to add that single function.

LDOS provides two commands to aid in interfacing drivers and filters. The SET command is used to define a new device or re-define an existing device and set that device to a driver. FILTER is used to interface a filtering routine to an existing device located in the Device Control Block tables.

The SET command takes the device spec (*XY) from the command line "SET *XY to DRIVER" and searches the device control block tables for a matching device. If the requested device is not defined in your configuration (use the DEVICE command to find out), SET establishes a device control block for the new device. In either case, control then passes to the DRIVER with register pair DE containing the address of the device code table for the "SET" device This points to the TYPE code (see the section on Device Control Blocks for a detailed explanation of the TYPE byte).

Register pair HL points to the command line character separating the DRIVER filespec and optional parameters. This provides DRIVER with the opportunity of parsing a parameter string by using a parameter table and the @PARAM system vector. What differentiates FILTER from SET is that FILTER will abort and provide an error message "device not available" if the device is not defined - i.e. you can only FILTER an existing device. FILTER also provides a default file extension of /FLT while SET uses /DVR.

The SET and FILTER commands are designed such that the DRIVER or FILTER routine should be ORGed at address X'5200' and automatically relocate itself to high memory. The routine must load between X'5200' and X'59FF' to be non-destructive. HIGH$ must be properly set after your routine relocates. Samples of filters are provided which should demonstrate the technique of writing the "relocating driver" portion of your routine.

To properly place a FILTER, it must be between the device control block and the existing DRIVER software. This can be accomplished by stuffing the filter entry point into the device control block vector address.

But first we save the existing vector to use in our filter so that we can transfer control to the existing driver software after we filter the flow of I/O.

Additional checking can be performed depending on whether the filter is one directional or two directional. The ROM calls @GET, @PUT, and @CTL initialize the CARRY and ZERO flags to indicate the I/O direction before passing control to the routine vectored from the device control table. A driver/filter can thus be aware of the request - input vs. output - and act accordingly.

If you examine the TRAP filter assembly language program, you will note that the filter itself takes up little space in high memory. The bulk of the filter is a driver initialization routine. Although it may appear lengthy, its purpose is just to load the filter driver, insert the vectoring between the device control block and the existing device driver, and perform other maintenance functions. This filter also provides an option for specifying the character to filter at the DOS command level.

Ideally, a general purpose filter driver generator can be constructed which could introduce such benefits as code translation, adding of line feed after return, trapping of certain codes, etc.

```
                00010 ;TRAP/ASM - 11/01/80
0000            00020         TITLE   <SAMPLE FILTER>
                00030 ;*****
                00040 ;       Sample FILTER routine to demonstrate the
                00050 ;       use of the FILTER command in LDOS.
                00060 ;       This routine traps certain control codes
                00070 ;       from being sent to the device being filtered.
                00080 ;       Any output device can utilize this routine.
                00090 ;
                00100 ;       This routine also demonstrates the use of the
                00110 ;       parameter scanner in the operating system.
                00120 ;       A single byte to trap can be passed in the
                00130 ;       command line as a parameter. If not entered,
                00140 ;       it will default to X'0E', the infamous cursor
                00150 ;       on character which if sent to a printer, will
                00160 ;       cause expanded character mode on a lot of dot
                00170 ;       matrix printers if CURSOR ON is sent to *PR.
                00180 ;
                00190 ;       To filter the printer output, issue:
                00200 ;       FILTER *PR using TRAP (BYTE=X'dd')
                00210 ;
                00220 ;       Roy Soltoff - September 30, 1980
                00230 ;*****
000A            00240 LF      EQU     10
000D            00250 CR      EQU     13      ;<ENTER> key
402D            00260 @EXIT   EQU     402DH   ;DOS return entry
4030            00270 @ABORT  EQU     4030H   ;error abort
4049            00280 HIGH$   EQU     4411H   ;highest usable memory
4467            00290 @DSPLY  EQU     4467H   ;display message
4476            00300 @PARAM  EQU     4454H   ;parameter scanner
447B            00310 @LOGOT  EQU     428AH   ;display & log message
                00320 ;*****
                00330 ;       LDOS uses a SET & FILTER library command
                00340 ;       which loads at X'5A00'. This provides
                00350 ;       the opportunity to load all relocatable
                00360 ;       drivers initially at x'5200'. The driver loader
                00370 ;       should then relocate to high memory honoring
                00380 ;       the HIGH$ pointer & resetting it as needed.
                00390 ;*****
5200            00330         ORG    5200H
                00340 ;*****
                00350 ;       After processing the command line, the
                00360 ;       SET/FILTER command runs the driver. On
                00370 ;       entry to the driver, register pair DE
                00380 ;       contains the device code table address
                00390 ;       for the device identified in the command
                00400 ;       line The driver loader (initialization)
                00410 ;       follows.
                00420 ;*****
5200 1A         00500 TRAP    LD      A,(DE)          ;get device type
5201 E602       00510         AND     2               ;make sure its an
5203 2849       00520         JR      Z,NOGOOD        ;output device
```

```
5205 D5       00530         PUSH    DE              ;save device DCB
5206 E5       00540         PUSH    HL              ;save command line ptr
5207 215752   00550         LD      HL,MSG          ;point to initialization
520A CD6744   00560         CALL    @DSPLY          ;message and display it
520D E1       00570         POP     HL              ;rcvr command line ptr
520E 11BA52   00580         LD      DE,PARMTBL      ;point to parm table
5211 CD5444   00590         CALL    @PARAM          ;get parms if any
5214 DDE1     00600         POP     IX              ;recover device DCB
5216 2031     00610         JR      NZ,PARRERR
5218 110E00   00620 BPARM   LD      DE,14           ;init to X'0E'
521B 7B       00630         LD      A,E             ;xfer to reg A
521C 32B152   00640         LD      (TRAPBYT+1),A   ;stuff in filter
521F 2A1144   00650         LD      HL,(HIGH$)      ;reduce HIGH$ by the
5222 011000   00660         LD      BC,LAST-START   ;length of this driver
5225 AF       00670         XOR     A               ;clear the carry flag
5226 ED42     00680         SBC     HL,BC           ;calculate new HIGH$
5228 221144   00690         LD      (HIGH$),HL      ;driver now protected
522B 23       00700         INC     HL              ;point HL at new START
522C DD7E01   00710         LD      A,(IX+1)        ;xfer orig DCB vector
522F 32B852   00720         LD      (ACCEPT+1),A    ;to driver CALL
5232 DD7E02   00730         LD      A,(IX+2)
5235 32B952   00740         LD      (ACCEPT+2),A
5238 F3       00750         DI                      ;interrupts off for now
5239 DD7501   00760         LD      (IX+1),L        ;update DCB vector
523C DD7402   00770         LD      (IX+2),H        ;to filter entry
523F EB       00780         EX      DE,HL           ;xfer new START to DE
5240 21AA52   00790         LD      HL,START        ;load address of driver
5243 EDB0     00800         LDIR                    ;move driver to top
5245 FB       00810         EI                      ;clock back on
5246 C32D40   00820         JP      @EXIT           ;return to DOS
              00830 ;*****
              00840         error abort
              00850 ;*****
5249 219B52   00860 PARMERR LD      HL,PRM.MSG
524C 1803     00870         JR      ERREXIT
524E 217B52   00880 NOGOOD  LD      HL,ERR.MSG
5251 CD8A42   00890 ERREXIT CALL    @LOGOT          ;log error message
5254 C33040   00900         JP      @ABORT          ;abort the request!
5257 53       00910 MSG     DM      'Sample filter to trap control codes',CR
     61 6D 70 6C 65 20 66 69
     6C 74 65 72 20 74 6F 20
     74 72 61 70 20 63 6F 6E
     74 72 6F 6C 20 63 6F 64
     65 73 0D
527B 54       00920 ERR.MSG DM      'This filter is for output only!',CR
     68 69 73 20 66 69 6C 74
     65 72 20 69 73 20 66 6F
     72 20 6F 75 74 70 75 74
     20 6F 6E 6C 79 21 0D
529B 42       00930 PRM.MSG DB      'Bad parameters',CR
     61 64 20 70 61 72 61 6D
     65 74 65 72 73 0D
              00940 ;*****
              00950 ; Actual FILTER routine to shift up to HIGH$
```

```
                00960 ;*****
52AA 3805       00970 START   JR      C,GETBYT        ;jump if *GET request
52AC 2800       00980         JR      Z,PUTBYT        ;jump if *PUT request
                00990 ;*****
                01000 ;       if carry is not set and Z-flag is not set,
                01001 ;       then the request came from @CTL. This
                01002 ;       routine could just eliminate the "JR Z,PUTBYT"
                01003 ;       since it is shown strictly for demonstration
                01004 ;       of how to differentiate the three vectors.
                01005 ;*****
52AE 79         01010 PUTBYT  LD      A,C             ;grab the output byte
                01020 ;*****
                01030 ;       any coding for the entrapment of specific
                01040 ;       bytes can be done here
                01050 ;*****
52AF F5         01060         PUSH    AF              ;save the flag value
52B0 FE00       01070 TRAPBYT CP      00              ;space for trap char
52B2 2002       01080         JR      NZ,$+4          ;branch if not trapped
52B4 F1         01090         POP     AF              ;rcvr flag value
52B5 C9         01100         RET                     ;return if trapped!
52B6 F1         01110         POP     AF              ;rcvr flag value
52B7 C30000     01120 ACCEPT  JP      0               ;output to orig device
52B7            01121 GETBYT  EQU     ACCEPT          ;or input w/o filtering
52BA            01130 LAST    EQU     $
                01140 ;*****
                01150 ;      parameter table - format as follows:
                01160 ; 1 => 6-character parm word buffered with spaces
                01170 ; 2 => address of word to receive the value parsed
                01180 ; 3 => repeat 1 & 2 for as many parms desired
                01190 ; 4 => end with an X'00' to indicate the table end
                01200 ;
                01210 ;       The parameter scanner accepts parms in the
                01220 ;       following format:
                01230 ;       PARM=X'dddd' :=:hexadecimal entry (max 4-digits)
                01240 ;       PARM=ddddd   :=:decimal entry (max 65535)
                01250 ;       PARM="string":=:string entry, word address
                01260 ;                         will contain the address of
                01270 ;                         the 1st character of "string"
                01280 ;
                01290 ;       On return, PARAM will set the Z-flag if parsing
                01300 ;       is OK, else the Z-flag will be reset (NZ)
                01310 ;*****
52BA 42         01320 PARMTBL DB      'BYTE  '        ;parameter word
     59 54 45 20 20
52C0 1952       01330         DW      BPARM+1         ;storage address
52C2 00         01340         NOP                     ;table end indicator
5200            01350         END     TRAP
00000 Total errors


@ABORT          4030 @DSPLY          4467 @EXIT          402D
@LOGOT          428A @PARAM          4454 ACCEPT         52B7
@PARM           5218 CR              000D ERR.MSG        527B
ERREXIT         5251 GETBYT          52B7 HIGH$          4411
```

LAST	52BA	LF	000A	MSG	5257
NOGOOD	524E	PARMERR	5249	PARMTBL	52BA
PRM.MSG	529B	PUTBYT	52AE	START	52AA
TRAP	5200	TRAPBYT	52B0		

```
                00010 ;LINEFEED/ASM - 12/08/80
0000            00020         TITLE   <LINEFEED FILTER>
                00030 ;*****
                00040 ;       FILTER routine to add a line feed after a
                00050 ;       carriage return for use with printers that
                00060 ;       need a specific line feed to function.
                00070 ;
                00080 ;       To filter the printer output, issue:
                00090 ;       FILTER *PR using LINEFEED
                00100 ;
                00110 ;       Roy Soltoff - October 8, 1980
                00120 ;*****
000a            00130 LF      EQU     10
000D            00140 CR      EQU     13      ;<ENTER> key
402D            00150 @EXIT   EQU     402DH   ;LDOS return entry
4030            00160 @ABORT  EQU     4030H   ;error abort
4049            00170 HIGH$   EQU     4411H   ;highest usable memory
4467            00180 @DSPLY  EQU     4467H   ;display message
447B            00190 @LOGOT  EQU     428AH   ;display & log message
5200            00200         ORG     5200H
5200 1A         00210 ENTRY   LD      A,(DE)          ;get device type
5201 E602       00220         AND     2               ;make sure its an
5203 2839       00230         JR      Z,NOGOOD        ;output device
5205 D5         00240         PUSH    DE              ;save device DCB
5206 214752     00250         LD      HL,MSG          ;point to initialization
5209 CD6744     00260         CALL    @DSPLY          ;message and display it
520C DDE1       00270         POP     IX              ;recover device DCB
520E 2A1144     00280         LD      HL,(HIGH$)      ;reduce HIGH$ by the
5211 010F00     00290         LD      BC,LAST-START   ;length of this driver
5214 AF         00300         XOR     A               ;clear the carry flag
5215 ED42       00310         SBC     HL,BC           ;calculate new HIGH$
5217 221144     00320         LD      (HIGH$),HL      ;driver now protected
521A 23         00330         INC     HL              ;point HL at new START
521B DD7E01     00340         LD      A,(IX+1)        ;xfer orig DCB vector
521E 329452     00350         LD      (PUTBYT+1),A    ;to driver CALL
5221 329E52     00360         LD      (GETBYT+1),A
5224 DD7E02     00370         LD      A,(IX+2)
5227 329552     00380         LD      (PUTBYT+2),A
522A 329F52     00390         LD      (GETBYT+2),A
522D F3         00400         DI                      ;not during update
522E DD7501     00410         LD      (IX+1),L        ;update DCB vector
5231 DD7402     00420         LD      (IX+2),H        ;to filter entry
5234 EB         00430         EX      DE,HL           ;xfer new START to DE
5235 219152     00440         LD      HL,START        ;load address of driver
5238 EDB0       00450         LDIR                    ;move driver to top
523A FB         00460         EI                      ;enable interrupts again
523B C32D40     00470         JP      @EXIT           ;return to LDOS Ready
                00480 ;*****
                00490 ;       error handling
                00500 ;*****
523E 217152     00510 NOGOOD  LD      HL,ERR.MSG
5241 CD8A42     00520         CALL    @LOGOT          ;log error message
```

```
5244 C33040   00530          JP      @ABORT          ;abort the request!
5247 0A       00540 MSG      DM      LF,'This filter will add a '
     54 68 69 73 20 66 69 6C
     74 65 72 20 77 69 6C 6C
     20 61 64 64 20 61 20
525F 6C       00550          DM      'line feed to <CR>',CR
     69 6E 65 20 66 65 65 64
     20 74 6F 20 3C 43 52 3E
5271 54       00560 ERR.MSG  DM      'This filter is for output only!',CR
     68 69 73 20 66 69 6C 74
     65 72 20 69 73 20 66 6F
     72 20 6F 75 74 70 75 74
     20 6F 6E 6C 79 21 0D
              00570 ;*****
              00580 ;       Actual FILTER routine to shift up to HIGH$
              00590 ;*****
5291 380A     00600 START    JR      C,GETBYT        ;jump on *GET request
5293 CD0000   00610 PUTBYT   CALL    0               ;output to orig device
5296 FE0D     00620          CP      CR              ;was char a <CR>?
5298 C0       00630          RET     NZ              ;go back if not
5299 0E0A     00640          LD      C,LF            ;else put out the LF
529B 18F4     00650          JR      START
529D C30000   00660 GETBYT   JP                      ;don't filter input
52A0          00670 LAST     EQU     $
5200          00680          END     ENTRY
00000 Total errors


@ABORT          4030 @DSPLY          4467 @EXIT          402D
@LOGOT          428A CR              000D ENTRY          5200
ERR.MSG         5271 GETBYT          529D HIGH$          4411
LAST            52A0 LF              000A MSG            5247
NOGOOD          523E PUTBYT          5293 START          5291
```

M E M O R Y M A P
==========

This memory map of the LDOS Disk Operating System is not necessarily a complete map of the entire system. Rather, it represents what is felt to be all of the system vectors that could reasonably be used by accomplished assembly language programmers. A few words of caution are in order. ALL RAM STORAGE ASSIGNMENTS ARE FOR VERSION 5.1.X MODEL III ONLY. Those system vectors prefixed with the AT SIGN "@", can be considered to be consistent from release to release. In addition, data storage assignments postfixed with a dollar sign "$" are also to be considered consistent from release to release. More detailed information concerning these assignments will be found in the SYSTEM ENTRY POINTS and RAM STORAGE Assignments. Most locations that differ from the Model I version of LDOS were moved for compatibility with Model III TRSDOS.

ADDRESS	LABEL	DESCRIPTION OF LOCATION
X'000B'	@WHERE	Vector to resolve relocation address
X'0013'	@GET	Input a byte from a logical device or a file
X'001B'	@PUT	Output a byte to a logical device or a file
X'0023'	@CTL	Output a control byte to a device or a file
X'002B'	@KBD	Scan the keyboard, and return the character
X'0033'	@DSP	Output a byte to the video display
X'003B'	@PRT	Output a byte to the printer
X'0040'	@KEYIN	Accept a line of input
X'0049'	@KEY	Input a byte from the keyboard
X'0060'	@PAUSE	Suspend program execution
X'3033'	@DATE	Get today's date - format (xx/xx/xx)
X'3036'	@TIME	Get time of day - format (xx:xx:xx)
X'4015'-X'401C'	KIDCB$	Keyboard DCB
X'401D'-X'4024'	DODCB$	Video DCB
X'4025'-X'402C'	PRDCB$	Printer DCB
X'402D'	@EXIT	Normal program exit and return to LDOS
X'4030'	@ABORT	Abnormal program exit and return to LDOS
X'4033'	@DVRHK	Device Driver hook from ROM for byte I/O
X'403D'	@ADTSK	Add an interrupt level task.
X'4040'	@RMTSK	Remove an interrupt level task
X'4043'	@RPTSK	Replace the currently executing task vector
X'4046'	@KLTSK	Remove the currently executing task
X'405D'-X'407C'	DBGSV$	DEBUG and SYSTEM storage area - DO NOT USE
X'4217'	TIME$	Contains time of day
X'421A'	DATE$	Contains the current date
X'421D'	@ICNFG	Initialize configuration

ADDRESS	LABEL	DESCRIPTION OF LOCATION
X'4220'	JDCB$	Storage area for DCB address during JCL execution
X'4222'	JRET$	Storage area for RET address during JCL execution
X'4225'	INBUF$	Buffer area of 64 bytes for user command input
X'4265'	JFCB$	JCL FCB during DO processing
X'4285'	@KITSK	Task processing during KBD scan
X'4288'	TIMER$	This is the 33.333 ms heartbeat
X'4289'	DFLAG$	The system device flag
X'428A'	@LOGOT	Display and log a message
X'428D'	@LOGER	Issue a log message
X'4290'	@CKDRV	Check for drive availability
X'4293'	@FNAME	Fetch file name/ext from the directory
X'4296'	@CMD	Accept a new command
X'4299'	@CMNDI	Entry to command interpreter
X'42A1'	SFCB$	FCB for loading system overlays
X'42B8'-X'42B9'	KISV$	Save KI DCB vector
X'42BA'-X'42BB'	DOSV$	Save DO DCB vector
X'42BC'-X'42BD'	PRSV$	Save PR DCB vector
X'42BE'-X'42BF'	KIJCL$	Save KIJCL DCB vector
X'42C2'-X'42C7'	JLDCB$	Joblog DCB
X'42C8'-X'42CD'	SIDCB$	Standard Input DCB
X'42CE'-X'42D3'	SODCB$	Standard Output DCB
X'42D4'-X'42D9'	S1DCB$	Spare DCB
X'42DA'-X'42DF'	S2DCB$	Spare DCB
X'42E0'-X'42E5'	S3DCB$	Spare DCB
X'42E6'-X'42EB'	S4DCB$	Spare DCB
X'42EC'-X'42FF'		Storage for 2 character device names
X'4300'-X'43FF'	SBUFF$	A 256 byte buffer for system disk I/O
X'4400'	EXTDBG$	Vector to extended DEBUG
X'4402'	@MSG	Message line handler
X'4405'	@DBGHK	Used with DEBUG (do not use)
X'4409'	@ERROR	Entry to post an error message
X'440D'	@DEBUG	Enter the debugging package
X'4411'-X'4412'	HIGH$	Contains the highest unused RAM address
X'4414'	OVRLY$	Current system overlay resident
X'4419'	@DODIR	Do a directory display/buffer
X'441C'	@FSPEC	Fetch a file or device specification
X'441F'	OSVER$	Contains the operating system version number
X'4420'	@INIT	Open or initialize a file or device
X'4423'	PDRIV$	Currently accessed drive - physical address (1, 2, 4, or 8)

ADDRESS	LABEL	DESCRIPTION OF LOCATION
X'4424'	@OPEN	Open an existing file or device
X'4427'	LDRIV$	Currently accessed drive - logical number (0-7)
X'4428'	@CLOSE	Close a file or device
X'442B'	SFLAG$	System bit flag
X'442C'	@KILL	Kill a file or device
X'4430'	@LOAD	Load a program file
X'4433'	@RUN	Load and execute a program file
X'4436'	@READ	Read a record from a file
X'4439'	@WRITE	Write a record to a file
X'443C'	@VER	Write then verify a record to a file
X'443F'	@REW	Rewind a file to its beginning
X'4442'	@POSN	Position a file to a logical record
X'4445'	@BKSP	Backspace one logical record
X'4448'	@PEOF	Position to the end-of-file
X'444B'	@FEXT	Set up a default file extension
X'444E'	MULT	Multiply HL by A
X'4451'	DIVIDE	Divide HL by A
X'4454'	@PARAM	Parse an optional parameter string
X'4458'	@CKEOF	Check for end of file
X'445B'	@WEOF	Write end-of-file
X'445E'	@RREAD	Reread the current sector
X'4461'	@RWRIT	Rewrite the current sector
X'4464'	@SKIP	Skip the next record
X'4467'	@DSPLY	Display a message line
X'446A'	@PRINT	Print a message line
X'446D'	@LOC	Calculate the current logical record number
X'4470'	@LOF	Calculate the EOF logical record number
X'4473'	INTIM$	Contains an image of the interrupt latch
X'4475'	INTVCT$	This area contains eight vectors - one for each bit of the interrupt latch
X'4485'-X'44A4'	CFCB$	File control block for commands
X'4500'-X'4517'	TCB$	Interrupt Task Table
X'4700'-X'474F'	DCT$	Area reserved for the Drive Code Table
X'4754'	SELECT	Select new drive
X'4759'	TSTBSY	Test if requested drive is busy
X'475E'	SEEK	Seek a cylinder

ADDRESS	LABEL	DESCRIPTION OF LOCATION
X'4763'	WRSEC	Write sector
X'4768'	WRSYS	Write system sector
X'476D'	WRCYL	Write a cylinder
X'4772'	VERSEC	Verify a sector
X'4777'	RDSEC	Read a sector
X'478F'	GETDCT	Get Drive Code Table address
X'479C'	DCTBYT	Get a DCT field
X'4B10'	DIRRD	Directory record read
X'4B1F'	DIRWR	Directory record write
X'4B45'	RDSSEC	Read a SYSTEM sector
X'4B64'	DIRCYL	Get the directory cylinder number
X'4B6B'	MULTEA	Multiply E by A
X'4B7A'	DIVEA	Divide E by A
X'4DFE'-X'4DFF'	USTOR$	Points to an 8 byte user storage area
X'4E00'-X'51FF'		LDOS Overlay Area

R A M S T O R A G E A S S I G N M E N T S

I/O Control Blocks

DCT$ (Area = X'4700'-X'474F')

Area reserved for the drive code table. Each drive occupies ten table bytes. Specific data on each 10-byte area is discussed in the technical section entitled Drive Code Table.

KIDCB$ (X'4015'-X'401C')

Keyboard Device Control Block

DCB+0 .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... System use
DCB+4 .... System use
DCB+5 .... System use
DCB+6 .... System use
DCB+7 .... Cursor blink switch, 0= solid, 1= blink

DODCB$ (X'401D'-X'4024')

Video Display Device Control Block

DCB+0 .... Device type
DCB+1 .... Driver address, low-order
DCB+2 ... Driver address, high-order
DCB+3 .... Cursor position, low-order
DCB+4 .... Cursor position, high-order
DCB+5 .... Character at cursor position, if any
DCB+6 .... Cursor character
DCB+7 .... System use

PRDCB$ (X'4025'-X'402C')

Printer Device Control Block

DCB+0 .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... Physical maximum of lines per page
DCB+4 .... Counter of lines printed on current page
DCB+5 .... System use
DCB+6 .... System use
DCB+7 .... System use

JLDCB$ (X'42C2'-X'42C7')

Joblog Device Control Block

DCB+0 .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... Unused
DCB+4 .... Unused
DCB+5 .... Unused

SIDCB$ (X'42C8'-X'42CD')

Standard Input Device Control Block

DCB+0 .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... Unused
DCB+4 .... Unused
DCB+5 .... Unused

SODCB$ (X'42CE'-X142D3')

Standard Output Device Control Block

DCB+0 .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... Unused
DCB+4 .... Unused
DCB+5 .... Unused

S1DCB$ (X'42D4'-X'42D9')

First spare Device Control Block

S2DCB$ (X'42DA'-X'42DF')

Second spare Device Control Block

S3DCB$ (X'42E0'-X'42E5')

Third spare Device Control Block

S4DCB$ (X'42E6'-X'42EB')

Fourth spare Device Control Block

System Control Information

DATE$ (Address = X'421A'-X'421E')

Contains the current date

DATE$+0 .... Contains the two-digit year-80
DATE$+1 .... Contains the day of the month
DATE$+2 .... Contains the month
DATE$+3 .... Contains bits 0-7 of the day of the year
DATE$+4
- Bit 0 ..... Contains bit 8 of the day of the year
- Bits 1-3 .... Contain the day of the week
- Bits 4-6 .... Reserved
- Bit 7 ..... Set to "1" if leap year

DOSV$ (Area = X'42BA'-X'42BB')

Save area for *DO DCB jump vector address.

EXDBG$ (Area = X'4400'-X'4401')

Pointer to location in high memory of the extended DEBUGger.

HIGH$ (Address = X'4411'-X'4412')

Contains the highest unused RAM address

JDCB$ (Area = X'4220'-X'4221')

Storage area for DCB Address during JCL execution

JRET$ (Area = X'4222'-X'4223')

Storage area for RET Address during JCL execution

KISV$ (Area = X'42B8'-X'42B9')

Save area for *KI DCB jump vector address

KIJCL$ (Area =X'42BE'-X'42BF')

Save area for KIJCL DCB jump vector address

LDRV$ (Address = X'4427')

Currently accessed drive - logical number (0-7)

OSVER$ (Address = X'441F')

Contains the operating system version Number

OVRLY$ (Area = X'4414')

Contains the LDOS overlay currently resident in the overlay region.

PDRV$ (Address = X'4423')

Currently accessed drive - physical drive address (1, 2, 4, or 8)

PRSV$ (Area =X'42BC'-X'42BD')

Save area for *PR DCB jump vector address

TIMER$ (Address = X'4288')

This is the 33.333 millisecond heartbeat counter

TIME$ (Address = X'4217'-X'4219')

Contains the time-of-day

TIME$+0 .... Contains the seconds

TIME$+1 .... Contains the minutes

TIME$+2 .... Contains the hours

Interrupt Processor Task Vector Storage

INTIM$ (Address = X'4473')

Contains an image of the interrupt latch

INTVC$ (Address = X'4475'-X'4484')

This area contains eight vectors - one for each bit of the interrupt latch.

INTVC$+0 .... Vector for latch bit 0

INTVC$+2 .... Vector for latch bit 1

INTVC$+4 ... Vector for latch bit 2

INTVC$+6 .... Vector for latch bit 3

INTVC$+8 .... Vector for latch bit 4

INTVC$+10 .... Vector for latch bit 5

INTVC$+12 .... Vector for latch bit 6

INTVC$+14 .... Vector for latch bit 7

TCB$ (Address = X'4500'-X'4517')

This area contains the vector addresses for each of the twelve possible interrupt processor tasks executed by the real-time-clock assigned to INTVC$, Bit 7. Task slots zero through seven are executed at 266.667 millisecond intervals and are considered "low-priority" tasks. Task slots eight through eleven are executed at 33.333 millisecond intervals and are considered "high-priority" tasks.

TCB$+0 .... Task slot 0, currently unassigned.

TCB$+2 .... Task slot 1, currently unassigned.

TCB$+4 .... Task slot 2, currently unassigned.

TCB$+6 .... Task slot 3, assigned to the ALIVE function.

TCB$+8 .... Task slot 4, currently unassigned.

TCB$+10 .... Task slot 5, currently unassigned.

TCB$+12 .... Task slot 6, currently unassigned.

TCB$+14 .... Task slot 7, currently unassigned.

TCB$+16 .... Task slot 8, assigned to the LCOMM Communications Line scanning function.

TCB$+18 .... Task slot 9, assigned to the SPOOLer function and to the LCOMM printer despooling function.

TCB$+20 .... Task slot 10, assigned to the TYPE ahead function.

TCB$+22 .... Task slot 11, assigned to the TRACE function.

System Buffers

CFCB$ (Area = X'4485'-X'44A4')

File control block buffer area used during @CMD interpreting.

DBGSV$ (Area = X'405D'-X'407C')

Area used during DEBUG operation as a register save area and pointer save area. During non-DEBUG operation, this area is used by the SYSTEM and must not be disturbed.

INBUF$ (Area = X'4225'-X'4264')

Buffer area of 64 bytes for user command input. It contains the last command input by the user.

JFCB$ (Area = X'4265'-X4284')

Buffer area of 32 bytes for SYSTEM/JCL file control block during DO processing

SBUF$ (Area = X'4300'-X'43FF')

A 256-byte buffer for system disk I/o.

SFCB$ (Area = X'42A1'-X'42B4')

A 20-byte file control block used for loading system overlays

System Flags

SFLAG$ (Address = X'442B')

Bit 0 .... Set to a "1" if SVC table active
Bit 1 .... Set to "1" if RUNning an EXEC only file
Bit 2 .... Set to a "1" if LOAD called from RUN
Bit 3 .... "1" if SYSTEM (FAST), "0" if SYSTEM (SLOW)
Bit 4 .... "1" if <BREAK> disabled, "0" if <BREAK> enabled
Bit 5 .... "1" if DO is in effect, "0" if DO is not in effect
Bit 6 .... If set to "1", will force extended error messages
Bit 7 .... "1" if DEBUG is to be turned on, "0" if it is to remain off

DFLAG$ (Address = X'4289')

Bit 0 .... Set to "1" if SPOOLer is active in the system.
Bit 1 .... Set to "1" if TYPE ahead is active in the system.
Bit 2 .... Set to "1" if JKL is active in the system.
Bit 3 .... Set to "1" if PR/FLT is active in the system.
Bit 4 .... Set to "1" if KI/DVR is active.
Bit 5 .... Set to "1" if MiniDOS/FLT is active
Bit 6 .... Set to "1" if KSM/FLT is active.
Bit 7 .... Set to "1" if GRAPHIC is on in the system.

S U P E R V I S O R Y C A L L S

The LDOS supervisory call table (SVC table) provides an alternative method for accessing system routines and obtaining information about system status within assembly language programs. All of the services which LDOS provides to user programs can be requested by means of an SVC instead of a call to a fixed RAM address. This is intended to allow programs written for LDOS 5.1 to run on all future versions of LDOS without change, even if hardware or ROM differences between different computer models require system entry points and storage areas to be moved. An SVC is requested by loading the A register with the SVC number (in the range X'00'-X'7F') and performing a RST 28H instruction. Depending on the function requested, the other registers may be used to pass parameters or return results.

The following table lists the currently defined SVC numbers. More may be defined in future LDOS releases. Register usage is listed for those functions which differ from the corresponding direct call. Refer to the "Entry Points" section for more information on each call.

Please note that the SVC table is an optional feature in Model III LDOS 5.1 (refer to the SYSTEM library command.) LDOS 5.1 library commands and utilities do not use SVCs. A program which is written to use SVCs will not run unless the SVC table is in place. The first SVC call will cause an abort back to the "LDOS Ready" level with a SYS ERROR message.

Using the SVC table requires that KI/DVR be used. The system will not function properly if using the SVC table and the ROM keyboard driver.

DEC	HEX	LABEL	FUNCTION
0			Reserved for future use
1	01	@KEY	Scan keyboard, wait for character
2	02	@DSP	Display character at cursor, advance cursor Register <C> must contain character to display
3	03	@GET	Get one byte from a logical device
4	04	@PUT	Write one byte to a logical device Register <C> must contain the character to PUT
5	05	@CTL	Make a control request to a logical device Register <C> must contain the request
6	06	@PRT	Send character to the line printer Register <C> must contain the character to print
7	07	@WHERE	Locate origin of CALL
8			Reserved for future use
9	09	@KEYIN	Accept a line of input
10	0A	@DSPLY	Display a message line
11	0B	@LOGER	Issue a log message
12	0C	@LOGOT	Display and log a message
13	0D	@MSG	Message line handler
14	0E	@PRINT	Print a message line
15			Reserved for future use
16	10	@PAUSE	Suspend program execution
17	11	@PARAM	Parse an optional parameter string
18-20			Reserved for future use
21	15	@ABORT	Abnormal program exit and return to LDOS
22	16	@EXIT	Normal program exit and return to LDOS
23	17	@CMD	Accept a new command
24	18	@CMNDI	Entry to command interpreter
25			Reserved for future use

26	1A	@ERROR	Entry to post an error message
27	1B	@DEBUG	Enter the debugging package
28			Reserved for future use
29	1D	@ADTSK	Add an interrupt level task <DE> contains the TCB address, <C> contains the task number.
30	1E	@RMTSK	Remove an interrupt level task <C> contains the task number
31	1F	@RPTSK	Replace the currently executing task vector
32	20	@KLTSK	Remove the currently executing task
33			Reserved for future use
34	22	@DODIR	Do a directory display/buffer
35-40			Reserved for future use
41	29	SELECT	Select a new drive
42-45			Reserved for future use
46	2E	SEEK	Seek a cylinder
47	2F	TSTBSY	Test if requested drive is busy
48			Reserved for future use
49	31	RDSEC	Read a sector
50	32	VERSEC	Verify a sector
51-52			Reserved for future use
53	35	WRSEC	Write a sector
54	36	WRSYS	Write a system sector
55	37	WRCYL	Write a cylinder
56			Reserved for future use
57	39	@KILL	Kill a file or device
58	3A	@INIT	Open or initialize a file or device
59	3B	@OPEN	Open an existing file or device
60	3C	@CLOSE	Close a file or device

61	3D	@BKSP	Backspace one logical record
62	3E	@CKEOF	Check for end of file
63	3F	@LOC	Calculate the current logical record number
64	40	@LOF	Calculate the EOF logical record number
65	41	@PEOF	Position to the end of file
66	42	@POSN	Position a file to a logical record
67	43	@READ	Read a record from a file
68	44	@REW	Rewind a file to its beginning
69	45	@RREAD	Reread the current sector
70	46	@RWRIT	Rewrite the current sector
71			Reserved for future use
72	48	@SKIP	Skip the next record
73	49	@VER	Write then verify a record to a file
74	4A	@WEOF	Write end of file
75	4B	@WRITE	Write a record to a file
76-77			Reserved for future use
78	4E	@FSPEC	Fetch a file or device specification
79	4F	@FEXT	Set up a default file extension
80	50	@FNAME	Fetch file name/ext from directory
81	51	GETDCT	Get Drive Code Table address
82	52	DCTBYT	Get a DCT field <C> contains byte number (0-9)
83	53	DIRCYL	Get the directory cylinder number
84			Reserved for future use
85	55	RDSSEC	Read a SYSTEM sector
86			Reserved for future use
87	57	DIRRD	Directory record read

88	58	DIRWR	Directory record write
89			Reserved for future use
90	5A	MULTEA	8-bit by 8-bit unsigned integer multiplication <C> contains multiplicand, <E> contains multiplier
91	5B	MULT	16-bit by 8-bit unsigned integer multiplication <C> contains multiplier, <HL> contains multiplicand
92			Reserved for future use
93	5D	DIVEA	8-bit unsigned integer divide <C> contains divisor, <E> contains dividend
94	5E	DIVIDE	16-bit by 8-bit unsigned integer division <C> contains divisor, <HL> contains dividend
95-99			Reserved for future use
100	64	HIGH$	Contains the highest unused RAM address If HL = 0, then HL is loaded with current HIGH$ If HL <> 0, then HIGH$ is changed to (HL)
101-127			Reserved for future use

S Y S T E M O V E R L A Y S

A system as complex and flexible as LDOS would occupy considerable memory space to be able to provide all of its features. LDOS, however, makes extensive use of overlays in order to minimize the amount of memory reserved for system use. The compromise in using an overlay driven system is that while a user's application is in progress, certain disk file activities requested of the system may require the operating system to load different overlays to satisfy the request. This could cause the system to run slightly slower than a less sophisticated system which has more of its file access routines always resident in memory.

The use of overlays also requires that a SYSTEM diskette always be available in drive 0 - the system drive. Since the diskette containing the operating system and its utilities leaves little space available to the user, it is useful to be able to remove certain parts of the system software not needed while a particular application is running. In fact, you will discover that your day-to-day operations will only need a minimal LDOS configuration. The greater the number of system functions unnecessary for your application, the more space you can have available for a "working" system diskette.

The LDOS Utilities will not be discussed in this section. You should have enough information in the Utility section of the manual to determine which Utilities you will need on your working disk. The following will describe the functions performed by each system OVERLAY, identified in an LDOS DIR command (using the SYS parameter) by the file extension, SYS. If you need to remove a SYS file, use the LDOS PURGE command which can be used to purge system files without knowing the file's UPDATE password - you only have to know the disk's Master Password.

SYS0/SYS

This is not an overlay. It contains the resident part of the operating system (SYSRES). It is essential that any disk used for BOOTing the system must contain SYS0. It may be removed from disks not used for booting.

SYS1/SYS

This overlay contains the LDOS command interpreter, the routines for processing the @FEXT system vector, the routines for processing the @FSPEC system vector, and the routines for processing the @PARAM system vector. This overlay must be available on all SYSTEM disks.

SYS2/SYS

This overlay is used for opening or initializing disk files and logical devices. It also contains routines for checking the availability of a disk pack (services the @CKDRV system vector), and routines for hashing file

specifications and passwords This overlay must also reside on all SYSTEM disks.

SYS3/SYS

This overlay contains all of the system routines needed to close files and logical devices. It also contains the routines needed to service the @FNAME system vector. This overlay must not be eliminated.

SYS4/SYS

This system overlay contains the system error dictionary. It is needed to issue such messages as "File not found", "Directory read error", etc. If you decide to purge this overlay from your working SYSTEM diskette, all system errors will produce the error message, "SYS ERROR". It is recommended that you not eliminate this overlay, especially since it occupies only one granule of storage.

SYS5/SYS

This is the "ghost" debugger. It is needed if you have intentions of testing out machine language application software by using the LDOS DEBUG command. If your operation will not require this debugging tool, you may purge this overlay.

SYS6/SYS

This overlay contains all of the algorithms and routines necessary to service the LIBrary commands identified as "Library A" by the LIB command. This represents the primary library functions. Very limited use could be made of LDOS if this overlay is removed from your working SYSTEM disk.

SYS7/SYS

This overlay contains all of the algorithms and routines necessary to service the LIBrary commands identified as "Library B" by the LIB command. A great deal of use can be made of LDOS even without this overlay. It performs specialized functions that may not be needed in the operation of specific applications. Use the PURGE command to eliminate this overlay if you decide it is not needed on a working SYSTEM diskette.

SYS8/SYS

This overlay is needed to dynamically allocate file space used when writing files. It must be on your working SYSTEM diskettes.

SYS9/SYS

This overlay contains the routines necessary to service the EXTended debugging commands available after a DEBUG (EXT) is performed. This overlay may be purged if you will not need the extended debugging commands while running your application. In addition) if you purge SYS5/SYS, then keeping SYS9/SYS would serve no useful purpose.

SYS10/SYS

This system overlay contains the procedures necessary to service the request to KILL a file. SYS10 also contains the routines to service the @DODIR system vector. It should remain on your working SYSTEM diskettes.

SYS11/SYS

This overlay contains all of the procedures necessary to perform the Job Control Language execution phase. You may remove this overlay from your working disks if you do not intend to execute any JCL functions. If SYS6 has been purged (containing the DO command), keeping this overlay would serve no purpose.

G L O S S A R Y

The following terms are used throughout this manual, and are fully described here. All of the descriptions pertain to the user sections of this manual; most also pertain to the technical section.

abbr: The abbreviation for "abbreviation". It is used at the bottom of each "syntax" block and is followed by the allowable abbreviations for the parameters and switches involved with the command.

ASCII The alphanumeric representation of controls and characters as a single byte, falling within a range from 1 to 127 (sometimes including 0).

ASCII files Files generally containing only ASCII characters.

BACKGROUND TASK A job that the computer is doing that is not apparent to the user or does not require interaction with the user. Some examples are the REAL TIME CLOCK, the SPOOLer and the TRACE function.

BIT One eighth of a byte, one binary digit.

BUFFER An area in RAM that will temporarily hold information that is being passed between devices or programs.

BYTE The unit that represents one character to the TRS-80. It is composed of eight binary "bits" that are either ON (1) or OFF (0). One byte can represent a number from 0 to 255.

CONFIGURATION The status of the system and physical devices that are available to it. This configuration may be dynamically changed with several library commands and the SYSTEM command, and may be saved with a SYSGEN. If the system is SYSGENed, that configuration will be re-established each time the machine is re-BOOTed or re-started.

:d This is used to indicate that a drive spec (number) may be inserted where this is used. A drive spec must always be preceded immediately by a ":" as shown. If a drive spec is not to be given, then the ":" must not be used.

DCB Device Control Block, a small piece of memory used to control the status and the input and output of data between the system and the devices.

DEVICE A physical device located outside of the CPU (Central Processing Unit), whose purpose is to transmit/receive data to/from the operating system. The operating system is in total control of any activity directed to/from a DEVICE.

devspec The name associated with a device by which it is referenced. A "devspec" will ALWAYS consist of three characters; the first of which is an asterisk, followed by two upper case alphabetic characters.

DRIVER A machine language module used to control interactions between the operating system and a DEVICE.

***DO** An LDOS system device, the Video Display.

/ext The extension of a filespec. The use of /ext is sometimes optional. An extension (if used), must contain as its first character a "/" (slash), and may be followed by one to three alphanumeric characters.

FCB File Control Block, a small piece of memory used to control the status and the inputting and outputting of data between the operating system and disk files.

filespec The name by which a disk file is referenced. A "filespec" consists of four fields and two switches, of which the first field is always mandatory. A filespec is designated by the following format:

!filename/ext.password:d! - - where

"!" (preceding filename) is an optional switch. If this switch is set, filespec is taken to be absolute. This allows the accessing of a filespec that would otherwise be inaccessible (i.e. a filespec that is the same as an LDOS library command).

filename The mandatory name of the file

/ext The optional file extension

password The optional file password

:d The optional drive specification

"!" (following :d) is an optional switch. If this switch is set, the end of file marker for file (filespec) will be updated after every write to the file.

filename The mandatory name used to reference a disk file. A filename consists of one to eight alphanumeric characters, the first of which must be alphabetic.

FILTER A machine language routine which monitors and/or alters I/O that passes through it. "FILTER" is also the LIBrary command which establishes a FILTER routine.

/FIX The desired file extension for a PATCH file.

FOREGROUND TASK Jobs the computer does that are apparent to the user, such as running an applications program or a utility and interacting directly with the user.

GRAN An abbreviation used for the term GRANule. A GRAN is the minimum amount of storage used for a disk file. As files are extended, file allocation is increased in increments of GRANs.

I/O The abbreviation for Input/Output.

/JCL The desired file extension for a DO file. "JCL" is an abbreviation for Job Control Language.

***JL** An LDOS system device, the Joblog.

***KI** An LDOS system device, the Keyboard.

/KSM The desired file extension for a KSM file. "KSM" is an abbreviation for Key-Stroke Multiply.

LCOMM A sophisticated communications program capable of interacting with disk, printer, video, keyboard and the RS232 interface. LCOMM will dynamically buffer all the system devices. LCOMM is provided with the LDOS system.

LIBRARY A set of commands used to perform most operating system functions.

load module format A file format that loads directly to a specified RAM address.

LSB The Least Significant Byte in a hexadecimal word, sometimes referred to as the "low order byte".

MACRO Statements or verbs used in JCL.

MSB The Most Significant Byte in a hexadecimal word, sometimes referred to as the "high order byte".

NIL A setting of a device, indicating an inactive state. All I/O to/from this device will be ignored.

partspec An abbreviation representing "PARTial fileSPEC". A partspec may be used with certain LDOS LIBRARY commands in lieu of a filespec. A partspec may be composed of any combination of the four fields defining a filespec. No switches (i.e. the leading and trailing "!" in a filespec) may be contained in a partspec.

In addition, the filename and /ext fields of a partspec may be abbreviated with leading information and/or "wildcarded". Examples of partspecs are given in the LDOS manual where applicable.

-partspec Identical to a partspec, except that it is used as exclusion criteria during certain functions.

PARAMETER The information that follows a library command or a utility, on the command line. This information is passed to the job that will be executed to tell the job how you wish execution to take place. Parameters usually follow the command and are enclosed in parentheses.

parm The abbreviation for parameter described above.

password The password associated with a filespec, the use of which is optional. A password (if used), must contain as its first character a "." (period), and may be followed by one to eight alphanumeric characters, the first of which must be alphabetic.

PATCH A utility to make minor alterations to disk files.

***PR** An LDOS system device, the Line Printer.

RAM Random Access Memory. In the TRS-80, the free user memory in the Computer unit.

ROM Read Only Memory. In the TRS-80, the BASIC language and drivers stored in the Computer unit.

SECTOR A contiguous block of disk storage, defined to be 256 bytes, where each byte within the sector has an absolute location and byte identification number. All sectors have a predefined, absolute starting and ending location.

***SI** An LDOS system device, the Standard Input. It is not presently used by the LDOS system.

***SO** An LDOS system device, the Standard Output. It is not presently used by the LDOS system.

SWITCH A parameter with a definite setting, such as ON/OFF.

TOKEN A variable used in JCL.

/TXT The desired file extension for ASCII TeXT files.

UTILITY A program that provides a service to the user. Utility programs usually run "outside" of the operating system itself.

wc The abbreviation for WildCard. In LDOS, the absence of a field in a filespec during certain LDOS commands.

wcc The abbreviation for WildCard Character, <$> in LDOS.

WORD Two bytes in HEXadecimal format X'nnnn'. Usually entered in reverse notation: low byte, then high byte (LSB,MSB).

I N C A S E O F D I F F I C U L T Y

Your LDOS operating system was designed and tested to provide you with trouble free operation. If you do experience problems, there is a good chance that something other than the LDOS system is at fault. This section will discuss some of the most common user problems, and suggest general cures for these problems.

Problem 1) ... The system seems to access the wrong disk drives, or cannot read the diskettes.

There are two main causes of this problem. If you have special hardware, it must be configured properly with the SYSTEM (DRIVE=,DRIVER) command. Check the drive table display with the DEVICE command and make sure that it shows the correct drive configuration.

If you have trouble reading diskettes created on other operating systems, refer to the REPAIR Utility. That section will explain what is needed to make these types of disks readable.

Problem 2) ... RS-232 communications do not work, or function incorrectly.

If you experience RS-232 problems, the first thing you should do is to make sure both "ends" are operating with the same RS-232 parameters (baud rate, word length, stop bits, and parity). If these parameters are not the same at each end, the data sent and received will appear scrambled.

Some hardware, such as serial printers, require handshaking when running above a certain baud rate. It may be necessary to hook the hardware's handshake line (such as the BUSY line) to an appropriate RS-232 lead, such as CTS.

Problem 3) ... Random system crashes, re-occurring disk I/O errors, system lock up, and other random glitches-keep happening.

If you encounter these types of problems, the first thing to check is the cable connections between the TRS-80 and the peripherals. The contacts can oxidize, and this can cause many different random problems. Clean the edge card connectors on the CPU unit and the peripherals, and be sure all other cable connections are secure.

If you experience constant difficulty in disk read/write operations, chances are that the disk drive heads need cleaning. There are kits available to clean disk heads, or you may wish to have the disk drive serviced at a repair facility.

If you need to frequently clean the disk heads, you might be using some defective disk media. Check the diskettes for any obvious signs of flaking or excess wear, and dispose of any that appear even marginal. Tobacco smoke and other airborne contaminants can build up on disk heads, and can cause read/write problems. Disk drives in "dirty" locations may need to have their heads cleaned as often as once a week.

One common and often overlooked cause of random type problems is STATIC ELECTRICITY. In areas of low humidity, static electricity is present, even if actual static discharges are not felt by the computer operator. Be aware that static discharges can cause system glitches, as well as physically damage computer hardware and disk media.

If the system boots, but things continually go wrong from then on, hold down the <CLEAR> key during the boot and then re-configure the system.

C U S T O M E R S E R V I C E

The LDOS development and support team is committed to the needs of our customers. Note that support is available ONLY TO REGISTERED LDOS USERS, which means you must:

1) Complete and send in your registration form. This gives you 1 full year of support.

2) Each additional year of support is available for the then current annual support fee.

As a current, registered LDOS user the following services are available:

TOLL FREE 800 NUMBER for customer and dealer support. This number is only to be used for LDOS support and questions, but before you call, PLEASE FOLLOW THE STEPS IN THE NEXT SECTION (where the phone number is available).

SOFTWARE MAINTENANCE allows you to send, at any time, your ORIGINAL LDOS Master Diskette to LDOS Support with five dollars and receive a copy of the latest version of LDOS-5.1. The address is:

LDOS Support Services
c/o Galactic Software Ltd.
11520 N. Port Washington Rd.
Mequon, Wis. 53092

If you send in your Master Disk for an update, be sure it is properly packaged in a sturdy and bend resistant container. The container you send will be re-labeled and used to return your disk.

THE LDOS QUARTERLY NEWSLETTER will be sent to you, containing useful information from other LDOS users as well as technical information and editorial content from our development and support team.

MICRONET BULLETIN BOARD service sponsored by L.S.I. and available for you, if you are also a Micronet subscriber. The LDOS registration card contains a space for you to register your Micronet user number with our customer service department. You will then be logged into our bulletin board as a valid user. The bulletin board will be maintained by our customer service personnel on a regular basis, to answer questions and provide timely information.

NOTE - From time to time, the hours of our customer service department may vary, depending on our work load. If the hours have changed, the Quarterly newsletter will contain the new hours. The normal time period for 800 line service is from 10:00 AM to 12:00 noon, and from 4:00 to 6:00 PM, Central time.

IF YOU HAVE PROBLEMS

LDOS has been created as a powerful, flexible, and user-oriented system. If you do run into problems, before you pick up the phone, do this:

1) Read the IN CASE OF DIFFICULTY section and do the checks indicated there.

2) READ THE MANUAL.
   Check syntax and spelling carefully.
   Review notes and technical information.
   Verify your understanding of the purpose of the command.
   Check updates received or information in quarterly newsletters.

3) RETRY THE OPERATION.
   Repeat the procedure again.
   Reset (BOOT) and repeat the procedure again.
   Perform the same function in a different manner, if possible.
   (Let the Utility prompt for information rather than putting it in the command line or vice-versa, don't abbreviate the parameter, remove unnecessary system options, etc.)

4) THINK.
   Did it work last time? If so, - what has changed since then?
   Could it be a faulty diskette? Maybe another copy would work.
   Is everything turned on, plugged in, etc?
   Is a needed file not present on the disk, such as a system file, data file, etc?

5) WRITE IT DOWN!
   Make notes on the problem, the things you have tried, and the exact steps that led to the problem. The more detailed the notes, the better!

6) CALL.
   Call the following number during the proper time period, Monday through Friday, excluding holidays:

800-558-6901

IMPORTANT

Be sure to have your LDOS registration number handy - you will always be asked for it!

If your problem is technical or very detailed, it may be better to write our Customer Service Department. Include your complete phone number and registration number. We will look into your problem, and respond promptly.

LDOS LIMITED WARRANTY

Every effort has been made to assure the high quality and reliability of the LDOS product. With the purchase of LDOS the user is granted one year of support at no additional charge. This support shall be limited to the privilege of having the master disk updated as often as desired for the current update fee, the receipt of a newsletter for one year and the use of the toll-free LDOS SUPPORT line. To receive this support the user MUST fill out and return the registration card within 30 days of purchase. This support may be renewed for additional one year periods, at the then current annual support fee. Should a user find a valid error in the LDOS system, and clearly define it to LDOS SUPPORT, every effort will be made to correct the error. All support shall apply to currently registered LDOS owners only.

Logical Systems Incorporated and its associates on the LDOS product, assume no liability whatsoever, with regard to the reliability and/or fitness of the LDOS product for any application. All data and/or programs entrusted to the LDOS system and the computer that it is operating on are the sole responsibility of the user. Under no circumstances will L.S.I. or its associates be held liable for the loss of TIME, DATA, or PROGRAMS by the user. This warranty and support information refers to the LDOS 5.1.x product only.

FOR LDOS USER SUPPORT CALL.... 800-558-6901

CUSTOMER SERVICE

Q F B

This utility is designed to allow for a backup with format to be performed. Only floppy drives may be used, and the backup performed must be mirror image. The syntax is:

```
QFB :s :d (parm,parm,parm)

:s   is the Source drive. The colon is optional.
:d   is the Destination drive. The colon is optional.

The following optional parameters may be used:

ALL=    parameter used to specify whether all cylinders
        of the source disk will be read and copied to
        the destination disk, or only allocated
        cylinders will be used. The switch ON or OFF
        may be specified, with the default being OFF.

V1=     parameter used to specify whether or not a
        verify of the destination disk is to be
        performed on the 1st pass. The switch ON or OFF
        may be used, with the default being ON.

V2=     parameter used to specify whether or not a
        verify of the destination disk is to be
        performed on the 2nd pass. The switch ON or OFF
        may be used, with the default being OFF.

QUERY=  Query for parameters not specified. The switch
        ON or OFF may be used. The default is OFF

abbr:  ON=Y, OFF=N, QUERY=Q, ALL=A
```

The QFB (Quick Format and Backup) utility will allow for the creation of a mirror image backup of a source disk without having to format the destination disk prior to executing the backup. The normal means by which a mirror image backup is made using LDOS is to first format a diskette using the FORMAT utility, and then use the BACKUP utility to perform the backup. The limitations of the QFB utility are as follows:

1.) Two distinct floppy drives must be used.
2.) The source diskette must have been formatted using the LDOS 5.l.x FORMAT utility, and cannot contain any non-standard format.
3.) QFB will run exclusively on LDOS 5.l.x, versions 5.1.3 or later.

QFB will perform a "single pass" format and backup. If QFB is entered with no drives specified, prompts will appear for them. If drive numbers are specified, the first drive number will represent the source drive, and the destination drive will be the second drive number. If no parameters are specified, the defaults will be used.

Consider the results of entering the following command.

```
QFB 1 2
```

Drive 1 will be used as the source drive, while drive 2 will be the destination drive. Prior to QFB performing any action, a prompt will appear to load the diskettes. Once the proper diskettes have been installed, press <ENTER>, and the backup will begin. The following actions will take place.

1.) The source diskette will be logged in, to determine the type of format.
2.) Cylinder 0 of the destination diskette will be formatted.
3.) If cylinder 0 of the source disk contains data, it will be read into memory.
4.) If cylinder 0 of the source diskette contains data, the information stored in memory (see Step 3) will be written out to the destination diskette.
5.) Cylinder 0 of the destination diskette will be verified.
6.) Steps 2-5 will be repeated for all remaining cylinders.
7.) The following message will appear after the last cylinder has been verified:

```
Duplication complete     1 disk  created

Replace destination disks and press <ENTER> to repeat
 ..<R> to restart with new parameters
   ...or....<BREAK> to exit program.
```

8.) Press <ENTER> in response to this prompt to make another mirror image backup. Press <BREAK> to abort the QFB utility. The following prompt will appear:

```
Load SYSTEM diskette and hit <ENTER>
```

Place a system diskette in drive 0 and press <ENTER>, to return to LDOS Ready.

If it is desired to use QFB again with different parameters, press <R> in response to the prompt displayed in step 7. Doing so will cause the drives to be prompted for, and prompts will appear for all parameters.

If QFB is to be restarted, or the command QFB (Q=Y) is entered, the following prompts for the parameters will occur:

```
Duplicate unallocated tracks? (Y/N)
Verify on same pass? (Y/N)
Verify on second pass? (Y/N)
```

The first prompt relates to the ALL parameter. If it is answered with <V>, all cylinders will be read from the source diskette and written to the destination diskette, regardless of whether or not the cylinder contains information. If this prompt is answered <N>, only cylinders containing information will be read and written.

The next prompt relates to the V1 parameter. If it is answered with <V>, all cylinders on the destination diskette will be verified immediately after all writes. If answered <N>, no immediate verify will be done.

The final prompt corresponds to the V2 parameter. If it is answered with <V>, all cylinders on the destination diskette will be verified upon completion of all writing to the diskette. If answered <N>, there will be no second pass verification.

If an error occurs, an appropriate error message will be displayed, and a prompt will appear requesting the course of action that is desired. During any QFB operation, the <BREAK> key will be active, and can be used to abort the process.

IMPORTANT

QFB assumes that a mirror image backup is desired, and performs no check on the destination diskette with respect to the existence of data. Any existing information on a destination diskette will ALWAYS be destroyed. Also, QFB will NOT clear the Mod Flags of files on the source diskette.