GO:SYS™

Golden Oldies:
System Enhancement Package
OPERATOR MANUAL

GO:SYS™

Golden Oldies:
System Enhancement Package
OPERATOR MANUAL

Revision 1.0.0 11/17/88

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

Golden Oldies: System Enhancement Package

SOFTWARE LICENSE AGREEMENT

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

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

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

Golden Oldies: System Enhancement Package

Table of Contents

General Description

DOCONFIG: Generates or reloads system configuration files while at command level, during executing Job Control Language, or within a running BASIC program.

DOEDIT: Lets you edit video information; allows this edited information to be passed back through the *KI device.

KISTORE: A filter that will allow the simultaneous copying of all *KI keystrokes to a file or device.

MEMDIR: Generates a directory of low-memory system drivers and high-memory resident modules.

PaDS: Provides a technique for combining separately executable object programs into one file thereby saving directory slots and creating custom user libraries. This technique has been used for years on mainframe and mini computers.

PARMDIR: A Job Control Language generator and report writer using on-line disk directories as a data base. Its flexibility will assist you in creating extensive JCL files without having to type in reams of data.

SWAP: Provides the facility of reassigning logical-to-physical drive assignments already existing in the system's Drive Code Table.

WC: A "shell" processor that allows you to invoke compatible commands on a number of file specifications that match a wildcardspec entered on the command line.

ZSHELL: Allows you to temporarily redirect the input or output from or to any file or device instead of the normal *KI or *DO devices until the program being executed returns to "DOS Ready".

DOCONFIG

This program expands the power of the DOS "SYSGEN" command by giving you much greater control and flexibility in creating or restoring system configurations. The syntax is:

DOCONFIG filespec/CFG (Sysgen)	
filespec	is the file to save or restore. The file extension will default to "CFG" if omitted.
Sysgen	is specified to save a system configuration. If omitted, the system will be restored to the configuration of "filespec".
abbr:	SYSGEN=S

DOCONFIG is a major enhancement of the configuration capabilities of your DOS. DOCONFIG works in one of two ways. You can save the current configuration of your system to any file of your choice on any drive of your choice. You can also restore the machine's configuration at any time from any of the configuration files you created. The configuration file is constructed similar to the DOS CONFIG/SYS file (with minor exceptions), except that now you control configurations without having to re-boot your machine.

DOCONFIG can function from "DOS Ready" or from @CMNDR execution. It can also be executed from a Job Control Language file to either SAVE or RELOAD a configuration file while the JCL is executing. This will work even if a re-loaded configuration changes the drive assignment for the drive currently executing the JCL file - be it the system's SYSTEM/JCL file or your own execute-only JCL file. DOCONFIG is smart enough to correct the JCL interfacing being done by DOS if drive assignments are switched. If the JCL is SAVING a configuration, the CONFIG file will not reflect JCL as being active. This means that the resulting configuration can be reloaded without reentering the JCL that was currently executing. The use of DOCONFIG now gives JCL more power to run job streams that require revised high-memory configurations for selected applications. Wow, dynamic reconfiguration - on the fly!

You can even execute the DOCONFIG program for saving a configuration while running a BASIC program. This is achieved by executing the command as:

```
SYSTEM"RUN DOCONFIG filespec (S)"
```

Saving the state of the executing BASIC program will require a 52K configuration file. Restoring the saved configuration file by DOCONFIG at some future time will result in continuing the execution of the BASIC program at the statement following the <SYSTEM"RUN DOCONFIG ... >. The reloading of the configuration is normally done at DOS command level. This can also be done from within BASIC via another <SYSTEM"RUN DOCONFIG> command; however, your current BASIC state will be overwritten unless previously saved with a "DOCONFIG".

Please note that when saving a configuration, the standard DOS message:

```
User configuration built
```

will be displayed at the conclusion of saving the file. This is perfectly normal as DOCONFIG interfaces with and actually executes the DOS library module used to create a CONFIG/SYS configuration file.

A good reason to employ the power of DOCONFIG is to give you an easy means of swapping high-memory configurations - without having to re-boot your machine. Once you establish a particular configuration, say with COM/DVR, FORMS/FLT, KSM/FLT, ..., save it into a configuration file using DOCONFIG. You can easily restore your machine to that configuration with another DOCONFIG command. Flexibility is power!

Note: memory resident modules which connect to the DOS in non-standard ways may not be properly un-installed using DOCONFIG.

DOEDIT

The DOEDIT filter will allow the editing of video information and will allow this edited information to be passed back through the *KI device.

```
Set *devspec [to] DOEDIT/FLT (parm, parm...)
Filter *KI [using] *devspec
```

***devspec**	Device specification to which the filter is assigned. A recommended choice is to use "*DE".
Nocr=ddd	Character to inform DOEDIT not to send a Carriage Return after the edited text.
Cursor=ddd	Character to use as the cursor when DOEDIT is active.
Active=ddd	Keystroke to use as the edit activation character.
abbr:	Actively, Cursorily, Nocr=N

Note that all parameters may be entered in decimal (ddd), hexadecimal (X'nn'), or ASCII ("a") format.

Parameter Functions

NOCR=value

If this character is one position to the left of the cursor when <ENTER> is depressed, NO carriage return (X'0D') will be sent through the *KI device. This parameter defaults to 127 decimal (X'7F').

CURSOR=value

This character will be displayed as the cursor while in edit mode. It will not change your normal system cursor! This parameter defaults to 95 decimal (X'5F').

ACTIVE=value

Typing this keystroke will activate DOEDIT. This parameter defaults to <CLEAR-SHIFT><E>.

DOEDIT Keystroke Functions

<Up Arrow>. <Down Arrow>. <Left Arrow>. and <Right Arrow>

While DOEDIT is active these keystrokes will move the cursor one position up, down, left, or right, respectively.

<SHIFT><Left Arrow>

This key will move the cursor to the beginning of the current line.

<SHIFT><Right Arrow>

This key will move the cursor to the first space following the last non-blank character in the current line.

<BREAK>

This key will return control to the normal keyboard driver and passes NO characters through the *KI device. The cursor is restored to its position prior to activation of DOEDIT.

<CLEAR><Right Arrow>

This key will insert one space at the cursor location. All characters on the line to the right of the cursor shift one position to the right.

<CLEAR><Left Arrow>

This key will delete the character under the cursor. All characters on the line to the right of the cursor shift one position to the left.

<CLEAR><Space>

This key will redisplay the previously entered DOS command at the cursor position, without executing it, thus allowing editing of the command line.

<ENTER>

This key will act differently depending upon the presence or absence of the NOCR character, thus:

1) If the character immediately to the left of the cursor when <ENTER> is depressed is NOT the same as the NOCR character, then all characters to the left of the cursor on the current line will be sent through the *KI device, terminated with a carriage return (X'0D').

2) If the character immediately to the left of the cursor IS the NOCR character, then all characters to the left of the cursor on the current line (excluding the NOCR character itself) will be sent through the *KI device. The characters will NOT be terminated with a carriage return.

All other keystrokes in the range 32 through 127 (X'20'-X'7F') will overtype the character under the cursor, and the cursor will move one position to the right.

KISTORE

KISTORE is a filter that will allow the simultaneous copying of all keyboard (*Kl) keystrokes to a file or device. This filter will be useful in conjunction with ZSHELL's ability to retrieve standard input from a disk file. In order to install the filter, it is necessary to enter the commands:

```
SET *KR to KISTORE [(Rewind)]
FILTER *KI *KR
```

Rewind	Is used to force the storage file to be rewound to its beginning so that any contents of an existing file are ignored.
Abbr:	Rewind=R

Note: Parameters within brackets "[]" are optional.

Once KISTORE/FLT is established, the filter remains dormant until activated. Simultaneously depressing <CLEAR><SHIFT><O> will activate the filter. You must be using the LS-DOS keyboard driver to activate and deactivate the filter. KISTORE will save the contents of the last two lines of the video and display the prompt:

Filespec?

Enter the filespec that you want to use for the storage of the keystrokes. KISTORE will restore the contents of the last two video lines and begin the copying of keystrokes to the specified file. All keystrokes are appended to any already in the file (assuming the file was existing). This permits you to separate the storage into more than one session - with each session concatenating additional keystrokes to those already stored. If you want to reuse an existing file and ignore its contents, then specify the REWIND parameter when you install KISTORE.

The copying may be terminated by simultaneously depressing <CLEAR><SHIFT><X> and the file will be closed.

To use KISTORE to create a standard input file, run the application, depress <CLEAR><SHIFT><O> to open the storage file. Proceed to use the application as you would normally. When you are finished, exit to "LDOS

Ready" if applicable, and depress <CLEAR><SHIFT><X> to terminate the storage and close the file. The same application may then be executed later using the file as standard input and your computer will be able to duplicate all of your commands without you having to retype them.

MEMDIR

The MEMDIR program provides a directory of system low memory and high memory usage. It also provides status of bank-switched memory. MEMDIR may be invoked from "DOS Ready", from JCL, or from @CMNDR. From "DOS Ready", its syntax is:

```
MEMDIR (Print)

Print          send display to printer

abbr:          PRINT=P
```

Ever wonder what in the world was up in high memory when you execute a MEMORY command and it says HIGH$=X'E123'? Where did all that memory go? No need to wonder any more. MEMDIR is here to give you a directory of high memory. It tells you what program/module is there, where it resides, and how long it is. MEMDIR also shows you what is located in the system low-memory driver region. MEMDIR gives the information on your high/low memory usage in the following formatted display:

```
32K banks avail = xx/yy, In use = < ---------- >

Memory Directory HIGH$ = X'aaaa' Length = bbbb

module   Start Address      End Address       Length
name        X'cccc'           X'dddd'          eeee
```

xx = the number of memory banks currently free (includes bank 0);

yy = the total number of memory banks installed;

< --- > = Indicates "-" for free bank; "+" for bank in use;

aaaa = the address of the top of unprotected memory (or beginning of system low memory);

bbbb = the decimal length of protected memory in bytes (or the quantity of bytes used in system low memory);

name = the name of the module in protected memory;

cccc = the address of the first byte of the module;

dddd = the address of the last byte of the module;

eeee = the decimal length of the module in bytes.

MEMDIR makes use of the front end linkage header protocol as documented in the Technical Reference Manual for 6.x. For MEMDIR to work properly, all modules occupying protected memory must adhere to the system standard header.

If no memory is protected, MEMDIR will advise you of that fact. If a nonstandard module is encountered in protected memory, MEMDIR will attempt to locate the next properly headered module. The region occupied by the improperly headered module will be identified as "unknown".

MEMDIR will pause if it fills the screen with the directory; the display will subsequently pause after each screen page. Pressing any key will display the next page of the directory. The PRINT parameter will direct output to the printer in addition to the screen display.

Here's an example of such a memory directory listing:

```
MEMDIR 6.2.0 - Copyright 1985 MISOSYS, Inc.

32K banks avail = 06/11, In use = <----++++--->

Low Memory        Start = X'08FO'      Length    2512

Program        Start Address      End Address       Length
$KI               X'08F0'            X'OBB7'           664
$DO               X'OB88'            X'0E00'           633
$PR               X'0E01'            X'0E3C'            60
$FD               X'0E3D'            X'0FF3'           439
XLR8              X'0FF4'            X'106C'           121
$HD1              X'106D'            X'11B6'           330
PXM               X'11B7'            X'122A'           116
ERD               X'122B'            X'12BA'            96
PLO               X'128B'            X'12BF'            53
High Memory       HIGH$ = X'F2EB'      Length =   3348

Program        Start Address      End Address      Length

WAM               X'F2EC'            X'FE13'          2856
DOEDIT            X'FE14'            X'FFFF'           492
```

PaDS

General Information

Partitioned Data Sets (PDS) are not new to LS-DOS. The three library files, SYS6/SYS, SYS7/SYS, and SYS8/SYS, are PDS structures. Katzan, in *OPERATING SYSTEMS, A PRAGMATIC APPROACH,* defines a partitioned data set as "a data file that is divided into sequentially organized members." Katzan further states, "Each PDS includes a directory that points to the beginning of each member. Data sets of this type are most frequently used to store object programs - each member corresponds to a single object program. The PDS as a whole is referred to as a library. Operating system libraries and user libraries are stored in this fashion." This definition describes exactly, the two LIB files in LS-DOS.

The LIBrary PARaMeter (LIBPARM) table located in SYS1/SYS contains a SYSn code and an ISAM entry number used to identify both the proper SYS file containing the command's object code and its entry in the ISAM table directory located within the PDS SYS file itself. The LIBPARM table is used by the LS-DOS Command Interpreter which parses the command line and checks if the "filename" entered by the user matches up with a LIB command listed in the table. When a match is found, linkage is established with the system loader in SYSRES to denote the LIB file and the specific member entry satisfying the LS-DOS command entered. The member entry is denoted by the ISAM number assigned.

The system loader opens the SYSn LIB file and reads through the ISAM map table stored in the LIB file looking for a match to the ISAM entry number supplied by SYS1. This map table is a directory containing information relevant to each member in the file. Once the system loader finds the appropriate entry, it positions the file to the starting point of the member then loads and executes the module.

The PDS structure has provided a technique for combining separately executable object programs into one file thereby saving directory slots. It also saves time by not having to load an entire 10K-15K file just to get a few hundred bytes or a few thousand bytes of program loaded if all LIB commands were just one big file (as was the case in earlier systems). The overhead of having to read and search the member directory is minimal. This technique has been used for years on

mainframe and mini computers. LDOS and LS-DOS are the only known DOSs supporting PDS structures on the TRS-80.

Up until now, only the system library had this support. However, with this PaDS utility, you now have user PDS structures implemented.

The PaDS command can be used to create custom user libraries. A library could be a collection of a dozen utility programs - all stored under one name but directly executable by specifying the library name followed by the member name. Consider for a moment, that you have built a library containing PROCESS, DSMBLR, FED2, BINHEX, EDAS, and XREF. The library name MYLIB was chosen. You can then execute EDAS by entering:

```
MYLIB(EDAS)
```

at the LS-DOS ready prompt. If you wanted to build a custom LS-DOS command library, you could use PROCESS to extract DIR, COPY, KILL, DEBUG, ROUTE, and RESET from SYS6/SYS and SYS7/SYS and build them into a user SYSLIB. Then you could PURGE SYS6 and SYS7 which would save about 24K from your "custom" SYSTEM disk. When you want to do a directory, you would only need to type:

```
SYSLIB (DIR) :2 (A, I)
```

to achieve the same result as if you had typed DIR :2 (A,I) on a regular SYSTEM disk. Albeit you could have named your user library, "S" and save the entering of five characters each time you wanted to execute a member of the library. That would let you use "S(DIR)"!

Okay, what capabilities are included with PDS? The PDS command is itself a Partitioned Data Set. Members provided implement the following functions:

APPEND — Appends a new member or members to the existing PDS and update the member directory and map tables accordingly. Executable program members may have more than one entry point.

BUILD — Provides the capability of creating a new partitioned data set with a user designated maximum number of members. The PDS is composed of a Front End Loader program, a MEMBER directory, and an ISAM MAP table. Members are added via the PDS APPEND command.

COPY Transfers an image of a PDS member from the PDS to a designated file. The member will not be deleted from the PDS.

DIR Provides a directory of members listing the member name, member type, date of addition, and file space occupied.

KILL Makes a member inaccessible for access.

LIST Will list a specific member in standard hex format or ASCII format.

PURGE Removes killed member(s) from the PDS and compresses the file to reclaim the space previously occupied by the killed member(s).

RESTORE Restores a killed file to accessibility.

SQUEEZE Creates a new PDS by copying all active members from one PDS into the new one.

Command entry

The complete file specification for use with PDS files is:

```
FILENAME/EXT.PASSWORD:DRIVE(MEMBER)
```

A word of caution. The MEMBER specification is only valid when used with a PDS command. You cannot provide the memberspec in a file specification used with LS-DOS commands not specifically designed to function with Partitioned Data Set files. Use the memberspec only with the PDS command library!

Any PaDS executable program member may, of course, be executed directly at "LS-DOS Ready" or via the LS-DOS RUN command. When entering a PaDS

file specification for execution, any MEMBER name can be shortened to its minimum length necessary to uniquely match up with the member directory. For example, each PaDS library command begins with a distinct letter of the alphabet therefore, all PaDS library commands can be abbreviated down to one character and still retain their uniqueness. If two members were named COPY and CREATE, then the minimum length necessary to uniquely identify either would be a length of two (CO, CR).

Where member names are used within PaDS commands, such as PDS(KILL), or PDS(COPY), then the entire memberspec: must be entered. The command line entry abbreviation was chosen to minimize keystrokes where member names are frequently entered. The full memberspec is required within PaDS commands to maintain maximum integrity of the PaDS file and associated maintenance functions.

Procedures to create a PDS

The creation of a Partitioned Data Set is an easy task. To begin with, you should ask yourself what is it to be used for? Perhaps a collection of scores of utility programs - all combined into one larger file. A larger file will undoubtedly save disk space since small files will always use at least one granule of space (1.25K in 5-1/4" SDEN, 1.5K in 5-1/4" DDEN, 4K in 5-1/4" rigid, ...). Members of a PaDS use only that amount of space necessary to contain each member. The entire PaDS file uses space rounded to integral granules. So it makes good sense to concatenate your small /CMD files into one or more PaDS files. How many will you want to store? Use that estimate plus a few extra as the MEMBERS parameter in the PDS(BUILD) command to initialize a new file for use as a Partitioned Data Set.

Second, concatenate the /CMD files by using the PDS(APPEND) command. You could, of course, build a MAP file listing each addition to be made and use the MAP option of APPEND to streamline the process. Or use JCL to PDS(APPEND)! Then use the PaDS - it's simple to deal with.

Perhaps you would really like to be able to store a large number of small data files on a disk. Assume that you are using your machine for word processing in an environment where it is important to catalog letter files. Letters typically waste two to four sectors of storage space because file allocations are always made in granules. That can mean 500 to 1000 characters of storage space lost for each letter which can use up disk space fast. PDS(BUILD) a Partitioned Data Set to store 100-200 letters - each having a unique eight-character member name

with their own mini-directory. The PaDS file name can then become another field for use in cataloging the letters. PDS(BUILD) a couple of PaDS "letter" files for specific categories. Then a convention can be established for the member names that will assist in denoting their contents. Remember, the PDS(DIR) command produces a sorted directory to aid you. You will also be optimizing the storage space available on your "letter" disks providing storage space for more letters per diskette.

The following example will illustrate the steps required to initialize a ten-member Partitioned Data Set and concatenate five members - four of which are executable programs and one of which is a pure text documentation file.

```
PDS(BUILD) ULIB:2 (M=10)
PDS(APPEND) BINHEX ULIB.PDS:2
PDS(APPEN) DISCAT ULIB.PDS:2
PDS(APPE) MEMTEST ULIB.PDS:2
PDS(APP) FED ULIB.PDS:2
PDS(A) DOC/SCR:3 ULIB.PDS:2
```

Note that the member name, APPEND, can be abbreviated to as few a quantity of characters that still keep its name unique amongst all the members in the PaDS! The example chose to illustrate this convenience by denoting various acceptable forms for entering the member name. You can now execute, for instance, FED by entering:

```
ULIB(F) or ULIB(FE) or ULIB(FED) or ULIB(F
```

The closing parenthesis following the member name is required only when additional information is entered on the command line. Again note that the member name, FED, can be abbreviated because no other member name starts with the letter "F".

PDS(APPEND)

This PaDS library command is used to add new members to an existing Partitioned Data Set (PDS). The syntax is:

```
PDS(APPEND) filespec1 filespec2 (Map,Data)
PDS(A) filespec1. filespec2 (Map,Data)
```

filespec1	is the file you wish to add as a MEMBER, or is the name of a MAP file which contains the filespecs of the files to add.
filespec2	is the PaDS file to receive the MEMBER.
Map	indicates that filespec1 is a MAP file.
Data	forces the added MEMBER to be interpreted as a data file (in lieu of a program file).
abbr:	Map=M, Data=D

The APPEND command can be used to add either one or more members to the PaDS identified as "filespec2". Members may be executable programs (/CMD type files) or data files (anything which is not a /CMD type file). Any appending file with a file extension of "/CMD" which has as its first byte an X'01', X'05' or an X'1F' and has as its fourth from last byte an X'02' will be interpreted as an executable program file unless overridden by the DATA parameter. Any appending file not so identified will be interpreted as a data file. Since executable program files have the X'02' byte changed to an X'04' when stored in the partitioned data set, the DATA parm is supplied to restrict PDS(APPEND) from treating a data file, which matches the program file specification, as a program file. It should be quite rare to discover a data file which matches the executable program parameters.

A program member can have one or more entry points, each entry point identifiable as a distinct MEMBER name. Each entry point will require a separate PaDS directory entry; however, the module is stored only once in the PaDS. If multiple entry points are required, then you must prepare a MAP file which

is used to append the member to the PaDS. The MAP file data line provides the means by which multiple member names and entry points are identified to the PDS(APPEND) command.

A data file member cannot be executed by the operating system. Any attempt to do so will result in an immediate abort. Data file members can be catalogued for archival or other purposes and retrieved by the PDS(COPY) command.

It is a good idea to use a file extension of "/MAP" on a MAP data file. The reason will soon become evident. The format of a MAP file is as follows:

```
filespec1,member1,traadr1,member2,traadr2,...
filespec2,member1,traadr1,member2,traadr2,...
filespec3,member1,traadr1,...
.
.
.
```

As can be noted, each line contains the information for a single file which can have one or more member names and transfer addresses. The line is limited to 79 characters in length. Each line must be terminated with an <ENTER>. The comma "," is used to separate each field on a line. Also, spaces cannot appear in the line.

Both filespec1 and filespec2 will have their file extensions default to "/CMD". Therefore, if no extension is provided, "/CMD" will be assumed. If the MAP parameter is entered, then the default file extension for filespec1 will be "/MAP". This guards against inadvertently appending a MAP file to a PaDS by forgetting to enter the MAP parameter.

Example:

```
PDS(APPEND) BINHEX S.PDS:0
```

will add the member file, BINHEX/CMD, to the partitioned data set, S/CMD.PDS:0.

Informative Messages

Reading PDS member directory...

Will be displayed after the PaDS file has been opened and when the PaDS member directory is loaded into memory.

Technical note

Some users encounter a rare problem after appending a CMD-type program. The problem encountered is getting the error message, "Load file format error" when attempting to execute a CMD program that was appended to a PaDS. Where does this error come from and why? To answer these questions, a little background is in order.

When the PaDS utility was developed, providing a "library" function that used absolutely no high memory and was fast was the most important design goal. In order to accomplish the friendly member-name access procedure and provide the direct execution of members, PaDS uses a technique which capitalizes on the method in which the DOS loads and executes programs - as well as its own LIB members. Thus, each PaDS that is built by the PaDS utility, incorporates a program that executes when the PaDS is invoked. Thus, when you enter a command such as:

```
MYLIB(TEST)
```

the DOS executes a program called "MYLIB/CMD" and leaves register pair HL pointing to the left parenthesis when control is passed to MYLIB/CMD. In addition, register pair DE is still pointing to the system's File Control Block which was established to access that program. Recognizing that, the program which executes MYLIB/CMD can parse the command line and determine if a member specification was entered. Using the open FCB, it can also access any part of the MYLIB/CMD file.

The executing program is termed a "Front End Loader" (FEL) since it is in the front part of the PaDS and is used to load the requested member. The FEL is written to each PaDS that is built with the PaDS utility. The PaDS FEL loads into the library overlay region at X'2600'. Surprisingly, the FEL is less than 256 bytes. In order to keep it short, the error message feedback had to be kept to a minimum. Since messages can take up a lot of space, the FEL uses standard errors that reside in the DOS error dictionary. Therefore, if you try to execute a PaDS member that is not a CMD program (a data file member, for example),

instead of displaying a message such as *"Attempted to execute a data member"*, PaDS passes an error code 34 to @ERROR. Now error-34 is *"Load file format error"* which is actually correct for this case. A data file generally is not a load module! In using error-34, PaDS saves about 40 bytes.

Now you know what that *"Load file format error"* is all about. OK you say, you told me what it means; but I tried to execute a member that was a CMD program! What gives? Most likely, if you would do a PDS(D) of your PaDS, you would find that the member in question was listed as a DATA member. Since you know that you appended a CMD program, how then did it get classified as data? That's an easy one. In order to classify a file as either DATA or PROGRAM, the PDS(APPEND) module imposes a few tests. If the file extension is "/CMD" AND the first byte of the file is X'05', X'0F', or X'01' [which indicate a load module HEADER record, copyright record, or LOAD record respectively] AND the fourth from last byte is an X'02' [which indicates a TRANSFER record], then the file is interpreted as a PROGRAM; otherwise it is interpreted as data. The reason for the classification is that CMD programs must have the TRANSFER record changed from an X'02' to an X'04' (which indicates END of partitioned data set member) so that they may be properly loaded and executed by the DOS loader. We certainly don't want every member to have a byte changed.

Now you say that your "troublesome" CMD program executed properly from DOS but won't as a PaDS member. When DOS loads a plain-vanilla CMD program, it reads through the file and loads X'01' records into memory. The DOS reads through until it reaches the TRANSFER record. It doesn't care if anything is beyond the TRANSFER record [if it did, I would not have been able to implement the PaDS utility as I did]. Thus, you will find that the CMD program did not have a proper end-of-file pointer in the directory - proper meaning that it indicates the last byte of the TRANSFER record as the end of the file. Thus, when PaDS read the file, it could not detect the X'02'.

There is a simple solution to the dilemma. First, LIST the CMD file in hex and note where the listing ends. If you don't see as the last four bytes, "02 02 LL HH", where LL and HH refer to the low-order and high-order bytes of the transfer address, then the file's EOF is wrong. All that you need do is run PROCESS, load the CMD file, then write it back out. PROCESS reads a file until the TRANSFER record is reached. When PROCESS writes out its buffer, it will create a file with an EOF that is correct for the CMD file.

PDS(BUILD)

This PaDS library command is used to initialize a file for use as a partitioned data set. The syntax is:

PDS(BUILD) filespec (Members,Loader="filespec")	
PDS(B) filespec (Members,Loader="filespec")	
filespec	is the partitioned data set to be initialized. It will be created if not already existing.
Members	is the directory size in members. The PaDS will be created with a size of 16 if omitted. The parameter value must be in the range <1-255>.
Loader	is used to initialize the PaDS with a Front End Loader (FEL) module other than the standard FEL supplied with the PaDS utility.
abbr:	Members=M, Loader=L

The PDS(BUILD) command must be used to initialize a partitioned data set prior to adding members with the PDS(APPEND) command. A Front End Loader program as well as empty MEMBER and ISAM tables constituting the directory, are used to initialize the file for use as a partitioned data set.

A PaDS is an extremely important file since it can contain up to 255 members and each member is itself a file. It is essential, therefore, that you do not inadvertently kill, write to, or copy to a partitioned data set. In fact, the only time writing should be done to a PaDS is during a member addition via the PDS(APPEND) command. For this reason, BUILD will automatically assign password protection to a Partitioned Data Set during the building process. The following attributes will be assigned:

```
OWNER PASSWORD: PDS
PROTECTION LEVEL: READ
```

You may change the password to one of your selection by using the system ATTRIB command after the PaDS is built. It is strongly suggested that you NOT remove the protection so as to avoid inadvertent destruction of the PaDS. Remember, most of the accesses to a Partitioned Data Set will be in a READ mode and will not require entry of the password. The password would only be needed to append a member, KILL a member, PURGE all KILLed members, rename the entire PaDS, or KILL the entire PaDS. A Partitioned Data Set can contain a maximum of 255 members - the maximum sized directory. The maximum size is defined at initialization. If you do not enter a MEMBERS parameter, a value of 16 directory slots will be provided. A directory size chosen should depend on your drive capacity and file sizes to be included as members.

In order to provide automatic execution of an executable program file member, a Front End Loader (FEL) program is required in every Partitioned Data Set. A standard FEL is supplied within the PDS(BUILD) library command that supports the execution of program members as identified in this document. Where custom Partitioned Data Set structures are implemented, an appropriate FEL will be provided.

PDS(COPY)

This PaDS library command will copy a MEMBER from a partitioned data set (PDS) to a standard type file. It can be used to retrieve archived member data files. The syntax is:

```
PDS(COPY) filespec1(membername) filespec2
PDS(C) filespec1(membername) filespec2

There are no parameters
```

This command will transfer the image of a member identified as " membername" from the partitioned data set identified as "filespec1" to a file identified as "filespec2". Both filespecs will be assumed to have a file extension of "/CMD" unless one is provided.

If the member being copied was an executable program having multiple entry points, the transfer address provided for the destination file will be the entry point of the member name provided in the command line.

Examples:

```
PDS(COPY) MYLIB:5(EDAS) EDAS:1
```

will copy the member, EDAS, from the partitioned data set, MYLIB/CMD:5, to a file named EDAS/CMD:1.

PDS(DIR)

A PaDS library command to obtain a directory of partitioned data set MEMBERS. The syntax is:

PDS(DIR) filespec (K,P)
PDS(D) filespec (K,P)

filespec	is the PaDS file whose directory is listed.
K	is entered to list the member names of killed members in addition to the active members.
P	is entered to simultaneously direct the output to the *PR device.

The PDS(DIR) command will provide an alphabetized listing of all active and killed members in a PaDS. In addition to the member name, the listing will show either a "P" or "D" to indicate a program of data file member, the date that the member was added to the PaDS, and the length of the member in bytes. If the "K" parameter is entered, then killed members will be listed and noted by the presence of an asterisk following the member name. All active members will be listed first followed by the killed members.

Example:

```
PDS(DIR) PDS
```

```
PDS: PDS/CMD       11/18/88    Size:   10K Members:   9/ 10

append    P 17-Nov-88  1835    build     P 17-Nov-88    796
copy      P 17-Nov-88  1075    dir       P 17-Nov-88   1304
kill      P 17-Nov-88   552    list      P 17-Nov-88   1043
purge     P 17-Nov-88  1493    restore   P 17-Nov-88    602
squeeze   P 18-Nov-88  1443
```

PDS(KILL)

This PaDS library command is used to remove a PaDS member from the PaDS directory. The member is left intact until purged. The syntax is:

```
PDS(KILL) filespec(membername)
PDS(K) filespec(membername)

noarameters are required!
```

This command can be used to remove a member from the PaDS directory. Members killed cannot be executed or copied. Killed members will also not appear in a directory listing unless the "K" parameter is provided. A killed member is denoted in the directory by having the member name's first character high-order bit set to a one. Killed members may be restored to active status with the PDS(RESTORE) command unless the member has been purged.

Example:

```
PDS(KILL) MYLIB.PDS:1(JUNKFILE)
```

will make inactive the member, JUNKFILE, in the Partitioned Data Set, MYLIB/CMD.PDS:1.

PDS(LIST)

This PaDS library command can be used to display a Partitioned Data Set MEMBER in either ASCII mode or HEX mode in a manner similar to the system's LIST command. The syntax is:

PDS(LIST) filespec(membername) (Ascii,Print) **PDS(L) filespec(membername) (Ascii,Print)**
filespec(membername) is the PaDS member to be listed.
Ascii indicates an ASCII listing is desired
Print denotes that the output is to simultaneously be directed to the *PR device.
abbr: Ascii=A, Print=P

A Partitioned Data Set MEMBER can be listed to the *DO device and the *PR device by using this PaDS library command. The listing will normally be in standard HEX display unless the parameter, "ASCII", is entered. The ASCII display will not be line numbered.

The listing is normally presented to the *DO device. If you want to direct the output to a line printer, the PRINT parameter can be entered.

Example:

```
PDS(LIST) LETTERS/SCR:5(L1221005) (A)
```

will list the member, L1221005, from the LETTERS/SCR:5 Partitioned Data Set. The listing will be in ASCII.

PDS(PURGE)

The PaDS PURGE library command will physically remove killed members from a Partitioned Data Set. The storage space occupied by the purged members will be reclaimed. The syntax is:

```
PDS(PURGE) filespec (ALL)
PDS(P) filespec (ALL)

filespec          is the PaDS file to purge.

ALL               indicates that all killed members are
                  to be purged without prompting.

abbr:             none
```

This PaDS library command is used to eliminate dead space in a Partitioned Data Set occupied by "killed" files. The member name of each killed member will be identified to confirm its purge. You thus have the opportunity to selectively purge killed members. If the parameter, "ALL", is entered, then all killed members will be purged without the display of prompting messages.

The purge process is performed by bubbling up all members which follow the purged member. For large files which have a number of members to purge, this process can take a long time. You may wish to use the SQUEEZE utility to create a new PaDS file which contains only the active members from the PaDS file having members to purge.

PDS(RESTORE)

This PaDS command will restore a killed member that has not been purged. The syntax is:

`PDS(RESTORE) filespec(membername)`
`PDS(R) filespec(membername)`

`filespec(membername)` is the PaDS member to be restored.

There are no parameters

This PaDS library command will restore a member that has been killed from a Partitioned Data Set. If the member had been purged, no amount of "restoral" will reclaim the member.

Example:

```
PDS(R) MYLIB(IGOOFED)
```

will restore the member, IGOOFED, in the Partitioned Data Set designated as "MYLIB/CMD".

PDS(SQUEEZE)

The PaDS SQUEEZE library command will physically remove killed members from a Partitioned Data Set similar to PDS(PURGE); however, instead of deleting them from the original file, it creates a new PaDS file and copies all active members from the first file. The syntax is:

```
PDS(SQUEEZE) filespec newfile
PDS(S) filespec (ALL)
```

filespec	is the original PaDS file.
newfile	is the file to be created. If it exists, it will be overwritten.
abbr:	none

This PaDS library command is used to eliminate dead space in a Partitioned Data Set occupied by "killed" files by copying only the active members to a newly created file.

Example:

```
PDS(S) BIGLIB NEWLIB
```

will create a new PaDS library named "NEWLIB/CMD" and will copy all active members from the old "BIGLIB" library to the new one.

PARMDIR

This utility command is used to generate output based on conditional tests of directory information. The format of the output is completely under user control. PARMDIR can also directly generate a MAP file of data filespecs for use with PaDS, the Partitioned Data Set utility. The command syntax is:

PARMDIR partspec outputspec (parm,parm...)

partspec	is the partial file specification representing the class of files that you want to examine.
outputspec	is the file or device that is to receive the output. If no file extension is given, the default is /JCL. If the outputspec is omitted, its entry is prompted.
A, B, C	are the prefix positional parameters
X, Y, Z	are the postfix positional parameters
Inv, Sys, Mod, Date	have the same selection meanings as in the DIR command.
sOrt	The selected directory records will be in sorted alphabetic order, unless this parameter is turned off. Default is ON.
Enter	Allows you to change the default logical carriage return from the semicolon to another character. Default is ";" (semicolon).
Fspec	Each selected filespec will be written to the output filespec unless this parameter is suppressed (e.g. FSPEC=NO). Default is ON.

Parameters continued on the next page.

(Parameters continued)	
IF	This parameter allows additional tests to be made before a directory record is passed on to be processed.
Label	'This allows you to execute just a portion of a parameter library. If this parameter is used, the first record of the parmlib MUST be a label (i.e., "@PARM1", etc.)
Map	indicates that a PDS MAP data file is to be generated. Default is OFF.
Notes	Some JCL comment lines are written to the output device specification. One of them is your query, shown in JCL comment format. To suppress JCL comment lines, set NOTES=N. Default is ON.
Parms	This parameter points to the parm lib in which the rest of the query is located. The parameters in the file can be extended from the PARM DIR command line, but since the parmlib parameters are the last ones read, they can not be overridden. If only PARMS is entered without a filespec, you will be prompted for the file specification.
Video	Show the results of your PARMDIR query on the video screen, in addition to the output device spec. Default is OFF.
Abbr:	Date=D, Enter=E, Fspec=F, Inv=I, Label=L, Map=M, Notes=N. sOrt=O, Parms=P, Sys=S, Video=V
Note:	all parameters and keywords may be in upper or lower case.

PARMDIR –Parameterized Directory

Introduction

The PARMDIR utility will allow you to access the information in your disk directories and produce formatted reports, files, and even Job Control Language (JCL) files.

How to get started

It is simplest to get started in learning the power of PARMDIR by just typing in:

```
PARMDIR :0 *DO
```

You will note that the names of all of the visible files that you have on your :0 drive were sorted and written to the screen.

The first entry on the command line, after the PARMDIR command name, was entered as ":0". This partial filespec tells the program which directory records to look at. This is similar to the way that the DIR command works. You might have entered "/CMD:0" as the first parameter, and then the program would have shown you only the visible files with an extension of "/CMD" on the :0 drive. Another way might have been to use the "NOT" operator in the partial filespec, as in "-/DVR:0", to see all files except the drivers that were on your :0 disk. PARMDIR has no restrictions on the way that you construct the partial filespec. If the drive specification is omitted from the partial file specification, all drives will be searched. You may also use the dollar-sign character ($) to signify any character. The dollar-sign is sometimes called a wild-card character, since it can represent any character while performing a comparison. This can be used to search for similarly named files, selecting those filenames that agree with the partial filespec except for the positions where the wild-card character says that we don't care what character is in those positions. An example of this would be if you had a series of accounts-receivable files: AR0F010, AR1F010, AR2F020, AR2F010, etc. If you used the partial filespec "AR$F" you would see a listing of all of those files that are currently on-line (notice that the drive number was omitted from the partial file specification.

The second parameter represented where the output was directed to go. This can be a device specification like *PR or *DO (or even the *CL comm-line driver!), or it can be directed to a file. If a file is used without an extension, the default of /JCL will be automatically added to your filespec.

The real power of this utility lies in the third item following the utility name. This parameter string gives you the flexibility to produce a file that meets almost any of your needs.

Optional parameters

There are many optional parameters that have been developed to give you the ability to produce specialized directory listings and files. Some that you should already be familiar with from the DIR command, like INV to allow gathering directory records from files that have been marked invisible. Similarly, SYS will make the systems files available. Please be sure to note that files that have been marked SYS may include files that do not have the file extension of /SYS. The MOD parameter will only let the program look at files that have been modified since the disk's last backup. The output will normally be sorted in filename order, unless you enter "SORT=N" in the parameter string.

The DATE parameter has been designed to let you specify a date range or specific date that you wish to examine. To examine the directory records that had been updated on the 3rd of July, 1988, the DATE parameter would be coded as:

```
...,DATE="07/03/88",...
```

The range of dates from the 3rd to the 15th of July would be written as:

```
...,DATE="07/03/88-07/15/88",...
```

Notice that the dates are written with the opening date of the range given first and the last date following the hyphen (-). Both dates are not necessary. If the hyphen is present following a single date, PARMDIR assumes that all directory records written on or after that date are to be examined. If the hyphen precedes the date, all directory records written on or before that date are used. For example,

```
...,DATE="07/03/88-",...
```

would produce output for records dated 3 July, 1988 or later, while:

```
...,DATE="07/03/88",...
```

would show only those records written on, or before, the 3rd of July, 1988.

The letters A, B and C are prefix parameters, appearing in the output before the file specification, while the letters X. Y and Z are postfix parameters, appearing after the filespec. These prefix and postfix parameters can be used to generate substitution tokens, used in JCL processing. If you just enter "A" in the parameter string, for example, the output will show a #A# before the file specification. An X would generate an #X# token following the file spec. Let's see an example.

```
PARMDIR /CMD:0 DOFILE:1 (A,X)
```

will produce a JCL file that looks something like this.

```
. PARMDIR: /CMD:0 DOFILE:1 (A,X)
#A# DOCONFIG/CMD:0 #X#
#A# MEMDIR/CMD:0 #X#
#A# PARMDIR/CMD:0 #X#
#A# PDS/CMD:0 #X#
#A# U/CMD:0 #X#
//exit
```

This DO file could be used by executing

```
DO DOFILE/JCL:1 (A=LIST,X="(HEX)")
```

which would produce the SYSTEM/JCL file:

```
. PARMDIR: /CMD:0 DOFILE:1 (A,X)
LIST DOCONFIG/CMD:0 (HEX)
LIST MEMDIR/CMD: 0 (HEX)
LIST PARMDIR/CMD:0 (HEX)
LIST PDS/CMD:0 (HEX)
LIST U/CMD:0 (HEX)
//exit
```

Note the use of the double quotes surrounding '(HEX)' so that the left parenthesis is properly parsed. If the same JCL file had had the "X" token substituted with the "(P)" parameter to the LIST command, after the DO compile had been completed, the SYSTEM/JCL file would have looked like this.

```
. PARMDIR: /CMD:0 DOFILE:1 (A,X)
LIST DOCONFIG/CMD:0 (P)
LIST MEMDIR/CMD:0 (P)
LIST PARMIR/CMD:0 (P)
LIST PDS/CMD:0 (P)
LIST U/CMD:0 (P)
//exit
```

These positional parameters can be made to hold strings by typing the pre/postfix. parameter followed by an equal sign and the string information contained between double-quotes, e.g. A=" any string ". Let's try an example. Execute the following,

```
PARMDIR :0 *DO (A="LIST")
```

You will notice that the output directed to the screen by the *DO device specification will be a series of lines,

```
. PARMDIR :0 *DO (A="LIST ")
LIST COM/DVR:0
LIST COMM/CMD:0
LIST EDAS/CMD:0
LIST FORMS/FLT:0
LIST KSM/FLT:0
LIST PDS/CMD:0 //exit
```

The first line is a JCL execution-time comment that echoes the PARMDIR query, while the last line signals the end of a JCL stream. Notice that the lines in between append all of the visible file names from drive :0 to the word "LIST ". If this query had been written to a file, you could have typed in "DO = filename" and the JCL processor would have listed each one of these files.

The next optional parameter can be quite useful to those of you who use the PaDS partitioned data set utility. One of PaDS' commands, APPEND, can accept a list of files that are to be inserted into a PaDS file. Using the MAP option, a MAP file that can be used in the APPEND operation is automatically generated. The MAP parameter will assume that all of the selected file records are non-CMD files, since PARMDIR will not read through the files looking for the transfer address information.

Let's try another example. Let us assume that a BASIC Cross-Reference utility named "BREF" is available that operates on programs loaded into BASIC. You may have a need to get the variable cross-reference listing from a series of BASIC programs, so you start writing the PARMDIR query.

```
PARMDIR /BAS:1 FILE (A="LOAD "
```

That would produce a JCL file that would look like:

```
LOAD PROGRAM/BAS:1
LOAD PROG1/BAS:1
LOAD PROG2/BAS:1
```

Now BASIC's LOAD is a funny animal, because it expects a double-quote in front of the program name that it is to load. If we put another quote after the LOAD, it would not work, because PARMDIR is looking at the quotes as its property. This presents us with a problem. Also, if you could place a carriage-return after the filename, you would be able to generate the 'SYSTEM "BREF"'. Well, PARMDIR will allow this, but you must be a little sneaky. Any time that you need a carriage-return in your output, you may enter a semi-colon (;) inside the quote marks.

If you wish to pass a semi-colon through to the output, without having it translated into a carriage-return, you may use the ENTER parameter to change the default logical carriage-return to any other character that you don't need by placing the new logical carriage-return character in quotes after the parameter. The ENTER parm can be entered in one of three forms; ENTER=ddd, ENTER='hh' or ENTER="c", where <ddd> represents a three character decimal character from 0 to 255, 'hh' is a two digit hexadecimal character within apostrophe marks, and "c" is any single character within quotation marks.

The situation sometimes exists where a character must be passed through that has special significance to either PARMDIR or DO. This can be rectified by using (within the quotation marks following a positional parameter) a per-cent mark (%) followed by two hexadecimal numbers. You can thus enter any character that can't be entered directly. Any file that you create using this method MUST be sent through the JCL compilation phase, since that is where the transformation from hex code to actual character takes place. Remembering that a hex 22 is a double-quote, let's go back to our problem.

```
PARMDIR /BAS:1 FILE (A="LOAD%22",X=";SYSTEM%22BREF")
```

After execution of this command line, PARMDIR will produce a file that looks like this:

```
. PARMDIR /BAS:1 FILE (A="LOAD%22",X=";SYSTEM%22BREF")
LOAD"PROGRAM/BAS:1
SYSTEM"BREF
LOAD"PROG1/BAS:1
SYSTEM"BREF
LOAD"PROG2/BAS:1
SYSTEM"BREF

//exit
```

Keywords

The real power of PARMDIR is explained in this section. Until this point, you have only been able to attach character strings to the filename. PARMDIR has been made more useful by the ability to place keywords, representing data items from the directory, into the output. This will be useful in our next project. Using a little ingenuity, you can prepare a file that can then be used to make a diskette library listing. If a file specification is used as the output-spec, every time that PARMDIR is executed, the file will be overwritten, which will only allow us to prepare a single diskette directory. However, if we first route a device such as *PR to a file, and then execute PARMDIR using this device as output, we will be able to keep executing the program (perhaps using the "repeat last DOS command" function) and keep adding new directory records to the end of the file. Try this:

```
ROUTE *PR DIR/FIL:0
PARMDIR :1 *PR (A="$NAX/$EXX $DAT …
  … $VNM",FSPEC=N,VIDEO,NOTES=N)
```

Now, put another diskette in drive 1 and repeat the PARMDIR execution for each of your diskettes. RESET *PR and you should have a file like this.

```
CMDFILE /CMD 10-Dec-81 DATA01A
DATACOMM/KSM 27-Dec-81 DATA01A
FILESPLT/    28-Feb-82 DATA01A
BUGT1982/VC  11-Jun-82 DATA01B
CAT     /JCL 22-Apr-82 DATA01B
PARMDIR /CMD 21-Apr-82 PARMDIR
PARMDIR /DOC 15-Jun-82 PARMDIR
JL      /DVR 10-Dec-81 LDOSSOLE
KSM     /FLT 20-Mar-82 LDOSSOLE
MINIDOS /FLT 20-Mar-82 LDOSSOLE
```

This file (somewhat abbreviated) represents the results of four executions of PARMDIR. Notice that the file has the various filenames, extensions, file dates and the name of the diskette in neat columns. With a little BASIC program, use of a `SYSTEM "SORT (VAR7="NAME", LEN=LENGTH)` sorting utility, a listing can be made with all of the files in filename (or date) order. Never again will you have to wonder about the location of some piece of software or a file!

Look again at the optional parameter. Did you notice the parameters FSPEC=N,NOTES=N,VIDEO"? Since the output was intended as a data file, rather than a JCL file, the "NOTES=N" turns off the generation of the JCL "//EXIT" command and the comments. If the "FSPEC=N" were not used, each record would be followed by the complete filename. This automatic feature was used when the JCL file was generated that LISTed all of the files on a diskette. Finally, the "VIDEO" parameter was turned on, since the output was going to a file and we wished to also look at it on the screen.

Here is a table of all the allowable keywords:

```
  $DAT       date that the file was last
             written, in the format MM/DD/YY.
* $EOF       end of file offset (1 to 256)
* $FM        ending record number (0 to 65535)
  $EXT       file extension
  $EXX       file extension extended
* $LRL       logical record length (1 to 256)
  $NAM       file name
  $NAX       file name extended
* $PRO       protection level (0 to 7)
* $REC       number of records (based on LRL)
             (0 to 65535)
* $DRV       drive spec (0 to 7)
  $TIM       modification time (hh:mm)
  $VID       16-byte volume name and date
             ($VID is the same as $VNM
             concatenated with $VDT)
  $VNM       8-byte volume (disk or diskette)
             name
  $VDT       8-byte volume date (MM/DD/YY)

Keyword can be used in the "IF" parameter
```

The two extended fields, $NAX and $EXX, are filled out with blanks if their values are shorter than the eight characters allowed for the filename and the three characters for the file extension.

Parmlibs

If you have been trying the examples as we went along, you will have noticed that the command line for the PARMDIR command rapidly approaches the 79 character limit for keyboard entry. In order to get around this command line size

limitation, PARMDIR has the ability to get its parameters from a parameter library (parmlib).

If you wish to use the parmlib facility, enter the PARMS parameter in the optional parm list. You may enter the filename of the parmlib in string format (e.g. PARMS="MYPARMS",) or if you wish to be prompted for the filename of the library, just enter "PARMS". PARMDIR assumes "/PRM" as the default extension for parmlibs.

The parmlib can be constructed using the DOS BUILD command, the TED editor, or any available text processor. You must be certain to save your text file using an ASCII option if you are going to use a text processor as PARMDIR requires proper line-terminating carriage returns.

Parameters can be written in a relatively free fashion. Parms can be written one to a line, followed by a carriage return, or several parameters separated by commas can be written on a line. These parameters can be extended by other entries on the command line, but cannot be modified. An example of this is that you could leave "VIDEO" out of your parmlib entry, and then add it later in the command line by entering "PARMS,VIDEO,etc.". However, if you did have VIDEO in the parmlib entry, you can not turn it off on the command line by using a "VIDEO=N" parameter.

Since the DOS is device independent, it is possible to use PARMS="*KI" to enter the parameters from the keyboard. This can be extremely helpful while testing. The way to signal the end of file to PARMDIR is to enter a logical end of file character. This is usually done by holding down the <CONTROL-SHIFT-@> keys simultaneously. Consult your DOS manual for details.

Since the smallest unit of disk drive allocation is the granule and most parmlibs are much shorter than that, having a separate parmlib for each set of parameters can be very wasteful of disk space. PARMDIR has gotten around this problem by allowing several sets of parameters to be combined into one parmlib, therefore making much better use of disk space. When you build your parmlib, if you add a label as the first line of each set of parameters, you can have many different sets of parameters contained in one parmlib. A label is formed by the at-sign ("@") followed by up to 15 alphabetic and numeric characters, however a more practical limit should be eight characters following the at-sign. Here is a sample of a parameter library:

```
@wrtlist
a="$NAX $EXX $drv $dat $rec $LRL $eof"
if="$pro > read"
@DIR056
a="$nax $exx $drv $dat $rec $lrl"
if="$drv = 0 + $drv = 5 + $drv = 6",f=n
```

You should notice that PARMDIR is not sensitive to the case in which the parameters are written. The "IF" parameter is covered in the next section.

Conditional execution (IF)

Earlier, you learned that it is possible to selectively look at files, depending on the partial filespec. The "IF" parm further allows you to conditionally accept or reject files, based on logical comparisons within the "IF" parm. In the table accompanying the keyword section, each one of the keywords that have been marked with an asterisk can be used in comparisons with data elements from the directories.

The comparison operators are similar to the ones used in BASIC and Job Control Language. They are:

<	Less than
>	Greater than
=	Equal to
<>	Not equal to
<= or =<	Less than or equal to
=> or >=	Greater than or equal to
&	logical AND
+	logical OR
-	logical NOT

Each of the data items that can be used with the "IF" statement, except for $PRO (protection level), are numeric. $PRO can be tested for the actual character abbreviation representing the protection level (C=none, E=Exec, R=Read, U=Updt, W=Write, N=reName, M=reMove, or F=Full,) or the number (0-7) that stands for the access level as shown below. Only the first character of the word is significant.

```
0-FULL      1-MOVE      2-NAME      3-WRITE
4-UPDATE    5-READ      6-EXECUTE   7-CANT
```

The IF="expression" can be compound and spaces between tokens arc acceptable. Logical expressions can be connected as in JCL by using the symbols for AND, OR, and NOT (&, +, and -). An example of a compound "IF" expression is

```
IF=" $LRL <= 10 & $REC >= 60000"
```

This would take all files that met the partial filespec comparison and would additionally check to see that the file had a logical record length (LRL) less than or equal to 10 and also that there were 60,000 or more records in the file, before it would accept the directory record for eventual output.

In the PARMLIB section above, again look at the sample parmlib. It contains two sets of parameters, labelled WRTLIST and DIR056. Referring back to the list of keywords, let's look at these members of the parmlib. Both of them create the output format using the "A" prefix parameter, but the "IF" parameter causes different sets of filenames to be examined and selected from the on-line directories. WRTLIST will accept only those directory records that meet the partial filespec from the command line and also only those files whose protection status is NAME, MOVE or FULL (i.e., greater than WRIT.) DIR056 was written to output the formatted directory information only for those files that were contained on drives ($DRV) 0. 5 and 6. Obviously, DIR056 was written for someone that had more than four drives, but who wasn't interested in the files contained on drives 1, 2, 3, or 4.

Error processing

There are two error types in PARMDIR. Syntax error means that something is wrong with the conditional "IF" expression, while the Type mismatch error occurs when a value field is a string and the command parser expects a number. In case of an error, the optional parameters will be listed on one line with the error being highlighted on the next line by a line of dots with a dollar-sign ($) under the error.

NOTE: In the case of a compound "IF" statement using AND logic, the first FALSE term makes the entire statement FALSE, therefore the parser will stop examining the rest of the statement. Thus, it is possible for an error in syntax or type to be un-flagged if it occurs in a term subsequent to an AND-connected FALSE term. The PARMDIR output, however, will still be correct.

SWAP

This program expands the power of the DOS "SYSTEM (SYSTEM=d)" command by allowing you to reassign the logical-to-physical drive assignments for any two drives even if Job Control Language is under execution. The syntax is:

SWAP :s :d	
:s	is the drive specification of the SOURCE drive. It may be in the range [0-7 & <> :d].
:d	is the drive specification of the destination drive. It may be in the range [0-7 & <> :s].

This utility is used to reassign the relationships between a logical drive number (specifications 0 through 7) and the physical disk drives associated through the system's Drive Code Table (DCT). SWAP will function from "DOS Ready", from JCL, or from @CMNDR execution (i.e. SYSTEM "RUN SWAP :s :d"). SWAP is especially useful within Job Control Language files that are used to install particular system configurations.

If you attempt to swap the SYSTEM drive (drive 0), the replacement drive must contain a system disk. SWAP will prompt for a system disk if necessary and able; however, if SWAP is being executed from JCL, it will abort.

If JCL is executing from one of the drives being swapped, SWAP will reassign the logical drive number used for the JCL to match the reassignment.

An example of a swap command is:

```
SWAP :4 :1
```

which will exchange the physical drive associated with logical drive 4 with that associated with logical drive 1. Note that what was referenced as drive 4 is now referenced as drive I and vice versa.

WC

WC is a "shell" processor that allows you to invoke compatible commands on a number of file specifications that match a wildcardspec entered on the command line. You enter the command line once while the WC shell processor searches the designated disk drive(s) for files that match your wildcard specification. WC builds a Job Control Language file of your command line (minus the "WC") substituting each matching file specification for the wildcard specification on a separate command line. WC then automatically executes the JCL file. You can invoke a WC command with:

`WC COMMAND wildcardspec parameters`

`COMMAND`	Is any DOS command using the syntax "COMMAND filespec...".
`wildcardspec`	Is the wildcard specification used to match on-line disk files. The wildcard syntax is described below.
`parameters`	Is any additional entries needed for the "COMMAND".

The "wildcardspec" uses the file name and file extension as two distinct fields for matching purposes. If the drive specification is entered, WC will search that specific drive for all files matching the name-extension wildcard fields. If the drive specification is omitted, then all drives will be searched. Within each field, WC accepts two wild characters, "?" and "*". The question mark will match any character in that character position. The asterisk is used to match all trailing characters in the field. For example, "?SHELL/TXT:1" will match with ASHELL/TXT, BSHELL/TXT, etc. but ASHELL1/TXT will not match. A global match of all filespecs would be an entry of the form, "*/*"; whereas a match of all /CMD files would be an entry of the form, "*/CMD". If a minus sign, "-", precedes the filename field, WC will select files that do not match the wildcardspec. Any entered password will be used in the full file specifications generated by the selection process. Note that this wildcard syntax is different from the DOS partspec!

WC is quite useful to perform repetitive tasks on files whose file specifications are similarly constructed. For example, to list out all /TXT files on drive 1, you could use a WILDCARD entry of-.

```
WC LIST */TXT:1
```

What DOS commands are compatible? All of the following DOS commands are: APPEND, ATTRIB, LIST, LOAD, REMOVE, RENAME, RESET, and RUN. Other programs that expect a filespec on the command line are also "compatible".

A word of caution - WC uses the SYSTEM/JCL Job Control Language file; therefore, WC cannot be executed from JCL!

ZSHELL

Who has not come across a BASIC program that was loaded with "PRINT" statements while that brand new printer was just connected to the machine. Well, time to change all of those "PRINT" statements to "LPRINT". Perhaps you learned the old technique of poking certain values into memory locations said to be the "Video Device Control Block" and were thus able to "modify" your program easily. In any event, you progressed to purchasing DOS and discovered this wonderful ROUTE command. With it, you could "ROUTE *DO to *PR" and run that BASIC program with the PRINTs mysteriously changed to LPRINTs. All you had to do was to "RESET *DO" after your program completed and returned to "DOS Ready".

DOS was a generation ahead of the "poking". There is, however, a more modem technique of dealing with the problem of wanting program output to go somewhere other than where the program was originally intending it to go. Or, for that matter, input to a program. ZSHELL is that next step. We have provided three significant features in this product. ZSHELL and your DOS system take you to another level of convenience and flexibility with program input/output.

What is ZSHELL anyway?

Almost every program needs some type of input and produces some type of output. In most cases, the input is retrieved from the keyboard, or in DOS, the *KI device (the "KI" stands for Keyboard Input while the asterisk, "*", is prefixed to indicate a device specification). Any output is then displayed on the video, or the *DO device (Display Output). ZSHELL is a system enhancement which will allow you to temporarily redirect the input or output from or to any file or device instead of the normal *KI or *DO devices until the program being executed returns to "DOS Ready".

Every one of us has a program which gets some input from the keyboard and then displays its output on the video. What do you do if you want the output to go to the printer (*PR)? You'd probably sit down and rewrite the program to change all the PRINT statements into LPRINT statements. Well, that leaves you with two different programs that do almost the same thing, except that their output goes to two different places. Well somehow that just doesn't seem fair to have to have two programs, so you whip out your DOS manual and find the ROUTE command. Aha, I can write one program to do the computing and use

the ROUTE command to cause the output to go to the *DO or the *PR without having two different programs.

OK, that sounds good to me. But what happens when the program finishes, and everything that should be displayed on the screen is still going to the printer? Well you have run into one of the primary disadvantages to using the ROUTE command. To unroute the device you use the RESET command; however, the RESET command not only will undo the route, it will also remove any drivers or filters from the device. This inability to unlink the ROUTEing but leave the drivers and filters associated with the device unaltered (the device chain), can be overcome by using ZSHELL. ZSHELL can perform the same redirection but can limit it to the duration of the program's execution. After the program's execution has terminated, all of the redirection will be removed and all of the devices will be left as they were before the redirection was done.

The process of causing keyboard input or video output to go to somewhere other than is normally expected (*KI or *DO) is called "I/O redirection". For the remainder of these instructions the *KI will be called the standard input and the *DO will be called the standard output. With ZSHELL, you can tell your computer where to get the standard input or where to put the standard output or both at the same time.

Now that you have an understanding of I/O redirection, we will advance to the next major capability of ZSHELL, piping.

You don't have to be a professional plumber to know that you have pipes running through your house or apartment. These pipes are used to distribute the water throughout your dwelling. ZSHELL, like your house, uses pipes, but instead of carrying water they carry information. With ZSHELL it is possible to cause the standard output of one program to become the input of another program. The information is said to be "piped" between the programs. Just visualize ZSHELL as an imaginary pipe carrying the information from one program to another.

ZSHELL's last ability is quite different from the first two and much easier to understand. ZSHELL allows you to enter more than one command on a command line - all you have to do is separate the commands with a semicolon. The first command will be executed. When it has completed, the next command will be executed. The only limit to the number of commands is the length of the command line - which can be up to 255 characters in length with ZSHELL.

What can I do with ZSHELL?

Well now that you have a basic understanding of I/O redirection and piping just how would you go about using them? To help you to understand what you can do, let's look at some examples.

Example# 1

The personal finance program you just completed, writes all of its output onto the video but you would like to have a copy appear on your printer. Instead of rewriting your program you run your program and instruct ZSHELL to redirect your output to the printer instead of the video.

Example # 2

You have a program which takes all its input from the keyboard and you would like to use it under JCL. After reading the DOS manual you find that since the program uses INKEY$ to scan the keyboard, JCL won't function. You can still accomplish what you want by making a file of the necessary keystrokes, run the program and instruct ZSHELL to route all standard input from that file.

Example # 3

You are trying to load a document which has tab characters (ASCII 9) into a text editor which doesn't "understand" the tab characters. You then remember that the LIST command has an option to expand tabs. Instruct ZSHELL to have the LIST command list the file with tabs expanded and pipe the output into the editor.

Executing ZSHELL

ZSHELL is a module that interfaces with the resident portion of DOS. ZSHELL loads and relocates itself to high memory. It protects itself by lowering the HIGH$ contents. Thus, before you can use the additional functions present in ZSHELL, you have to "install" it in high memory. This is done at "DOS Ready" by entering the command:

ZSHELL [(Drive=d,BREAK=sw,LENGTH=n,DISable)]

Drive=d	Place piping files on drive d with d being a valid drivespec <0-7>.
BREAK=sw	An option to send or omit a BREAK character upon reaching the end of the input file. Default is OFF
LENGTH=n	Establishes the length of the SHELL command line [255 max]. It defaults to the DOS command line length.
OFF	Is used to disengage the resident ZSHELL module and attempt to reclaim the high memory it used.
Abbr:	Drive=D, DISable=DIS=OFF=N

Note: Parameters within brackets "[]" are optional.

Typing ZSHELL from DOS Ready will cause ZSHELL to load, relocate itself to high memory, and then become active. After ZSHELL has been activated it will remain tied into the system until the computer is rebooted. A "global reset" cannot remove ZSHELL from the system. This is because ZSHELL is connected to the resident system module in addition to the devices. RESET cannot undo this "connection". Likewise, DOCONFIG cannot un-install ZSHELL.

The parameter, DISABLE, will be discussed first. Since ZSHELL is tied into the system and cannot be removed by RESET, the OFF parameter serves this purpose. Once ZSHELL is activated, if you want to de-activate it, simply enter the command:

```
ZSHELL (OFF)
```

ZSHELL will unhook itself from the system. If no other module was placed into high memory after ZSHELL was installed, then ZSHELL will reset HIGH$ to the value that existed prior to its installation. Thus, the high memory space it used will be freed for subsequent use.

The concept of piping is normally implemented on larger computers by executing the "interconnected" programs simultaneously. This is known as "multiprocessing". With it, a channel of communications is established between the two programs - it is this channel that is termed the "pipe". Your computer environment does not support "multiprocessing". Therefore, piping has been implemented by chaining one program to another with the "standard output" of the first temporarily stored in a holding file until the first program completes its execution. The second program then uses this holding file as its "standard input". Since the second program could have standard output "piped" to a third program, a second holding file is needed. The parameter, DRIVE, will allow you to instruct ZSHELL on which drive to place these piping files. DRIVE will default to drive=0 if you omit the parameter.

The DRIVE parameter can be changed by simply reinstalling ZSHELL with a new parameter string. Any parameter not entered on the command line will be left unchanged. The new copy of ZSHELL will re-use the same space in memory if it finds itself already installed.

The BREAK parameter is associated with redirection of standard input. Its use will be described in that section.

The LENGTH parameter is used to establish the size of the command line buffer which ZSHELL maintains. This can be up to 255 characters in length. The minimum is the DOS limit (79 for LS-DOS 6.x). A longer line buffer permits the entry of complex piping commands as well as more individual commands connected with the ";" multiplier. However, the command line passed to the DOS after parsing by ZSHELL and extraction of redirection segments can not exceed the DOS limit.

Command line syntax

While active, ZSHELL monitors the command line entered in response to the "DOS Ready" prompt. If the first character of the command line is a double-quote, "", then the command line will be passed unaltered to the command

Interpreter; the double-quote character will be removed. This could be useful for those MC users wanting the application to invoke any redirection desired. In addition, ZSHELL will ignore a command line which begins with a period and pass the line unaltered to the command interpreter. A period is used by Job Control Language to designate a comment line.

There are several special characters that, when entered on the command line, will cause ZSHELL to take appropriate action. Let's take a look at each character and see what it does.

Redirect output: >, >+. >>, >>+

The first character is the greater-than symbol or right caret, ">". When the " >" symbol is detected on the command line, ZSHELL will cause the *DO standard output to be redirected to the devicespec or filespec which follows the greater-than symbol. If the caret is followed by the plus sign, "+", then the *PR device is considered to be the standard output device. For example:

```
LIB >*PR
```

will direct the display of the LIB command to a printer instead of the video display Alternatively,

```
DIR :0 (A,I,P) >+CATALOG/TXT
```

will direct the "printer" output of the DIRectory command to the CATALOG/TXT file.

If the ">" symbol is followed by another greater-than symbol, the output will be APPENDED to the devicespec or filespec. For example:

```
>>MAP/DAT FREE :1
```

will append the drive 1 free space map to the end of the file, MAP/DAT. Note that this example shows the redirection specification first. Actually, the redirection specification can be anywhere on the command line except the middle of a parameter string. Thus, the following are equivalent command statements:

```
>>MAP/DAT FREE :1
FREE >>MAP/DAT :1
FREE :1 >>MAP/DAT
```

Isn't that flexibility? Remember that the standard output will be assumed to be the *DO device. The *PR device can be easily used as "standard output" by simply adding a plus sign immediately after the right caret.

Redirect input: "<", "<#", "<@"

The second character is the less-than symbol or left caret, "<". The less-than symbol causes ZSHELL to redirect standard input. Programs that take input generally have a couple of different methods of noting when no more input is available. Some expect you to depress the <BREAK> key when you have completed your input (e.g. the LDOS BUILD library command). Other programs are looking for a specific sequence of characters to signify the end of input. In this case, if an end-of-file is reached, it is an error and the program should abort. There are also programs which do not expect to ever see an "end-of-file" condition from their "standard input" (i.e. BASIC). To satisfy these three classical ways of handling input terminating conditions, ZSHELL provides three forms of input redirection. All three forms cause standard input to be redirected according to your device or file specification but cause different things to occur if an end-of-file is reached.

If the "<" symbol is entered immediately followed by the devicespec or filespec, than the standard input will be retrieved from that file or device. When and if the end-of-file is reached, ZSHELL will automatically disengage the redirected input and restore the original *KI handling. Thus, the standard input is retrieved once again from the original *KI device.

If the "<" symbol is followed by a number sign or pound sign, "#", an end-of-file is handled somewhat differently. The control of the standard input will be returned to the *KI device as noted above; however, a <BREAK> (ASCII X'01') character will be sent through the standard input before control is returned to the *KI. The BREAK bit in the KFLAG$ will also be set (the KFLAG$ is documented in the technical section of your LDOS manual. If you do not program in assembly language, you need not bother with the KFLAG$). This BREAK will allow a program to detect the end-of-file condition by scanning for a break. The pound sign character, "#", should be easy to remember if you correlate "pounding" on something as "breaking" it.

The last form of standard input redirection is the "<" symbol followed by an symbol. If you had specified a command entry of the form:

```
BASIC RUN"BATCH" <@DATAFILE/TXT:2
```

and an end-of-file was reached, ZSHELL will abort the current program by transferring control to the @ABORT vector. If Job Control Language (JCL) was active, the JCL will abort. Use this form of redirection if your program should not expect to read to the end of a file and would, in fact, be an error if it did. Remember the "@" to indicate @ABORT since it is the first character of "@ABORT".

Pipe output (1) to input (2): "|", "|+"

The third character is the vertical brace, "|". The vertical brace may be generated by simultaneously depressing <CLEAR><SHIFT></>. The "|" will cause ZSHELL to "pipe" the display output of the first command (*DO output) to the input of the second command. There may be any number of commands that are separated by the vertical brace which continue to pipe from left-to-right. You are limited by the maximum size of the command line. For example, a command line of:

```
DIR :0 (A,I,S,N) | MSCRIPT
```

will pipe the output of the directory display into MSCRIPT for immediate word processing. It is important to observe the parameter, "N", and why it is specified. Remember that certain programs presenting data on the display screen may pause when the screen is filled. These programs are expecting some keyboard response (typically an <ENTER>) before continuing. Just because you have redirected the display output to a printer or disk file, it does not eliminate the need for such a response. Programs that present output pauses should have options (such as the "N" parameter in the DIR command) to allow non-stop output. If you observe that a particular program may have stopped its output, it may be because of this pause. You may want to "nudge" it along by depressing <ENTER> or some other response as dictated by the program you are running. Without ZSHELL, to achieve the identical results as the above example, you would have had to enter the sequence:

```
ROUTE *PR TEMP/TXT:0
DIR :0 (A,I,S,P)
RESET *PR
LSCRIPT <SHIFT><ENTER> L TEMP/TXT
```

to accomplish the same function more easily provided by ZSHELL. The *PR device output can be used for piping standard output instead of the *DO device

by immediately following the vertical bar with a plus sign, This is done just as easily as it was done for output redirection.

Multiple commands:

The last character is the semi-colon, ";". ZSHELL will allow the entering of more than one command on a single command line. Just separate the commands with a semi-colon and ZSHELL will handle the rest. For example,

```
LIST TEST1/ASM ; LIST TEST2/ASM; LIST TEST/CMD:1 (H)
```

In this example, note that spaces may be inserted either before or after the semicolon. ZSHELL will disregard any spaces surrounding the semicolon as well as the vertical bar (piping). Again be aware that the redirection specifications can occur anywhere on the command line except within any parameter string. ZSHELL ignores all characters found between a left parenthesis, "(", and its closing right parenthesis, ")". Parentheses are used to surround the parameter strings. Note that most versions of LDOS permit you to omit the closing right parenthesis. However, if you are going to follow your parameters with redirection specifications, you must close your parameters with the right parenthesis so ZSHELL can accept your specifications. For example:

```
\DIR (A,I,S,N >CATALOG/TXT:3
```

will not redirect the output of the DIRectory command since the parameters are not "closed". The command should be entered in one of the following forms:

```
DIR >CATALOG/TXT:3 (A,I,S,N
>CATALOG/TXT:3 DIR (A,I,S,N
DIR (A,I,S,N) >CATALOG/TXT:3
```

Let's see some more examples

In the first example, all output sent to standard output (PRINT statements) will be redirected into the file TEST/TXT. When BASIC is exited. the file will be closed and the standard output returned to normal.

```
BASIC >TEST/TXT:0
```

Next, all standard input will be retrieved from the file, INPUT/TXT. If the end-of-file is reached. then control will be returned to the *KI device.

```
BASIC <INPUT/TXT
```

In the third example, all standard input will be retrieved from the file, INPUT/TXT, and all output sent to the file, OUTPUT/TXT. If the end-of-file is reached then the abort exit to "LS-DOS Ready" will be taken.

```
BASIC <@INPUT/TXT >OUTPUT/TXT
```

The final example has information normally displayed by the DEVICE command "piped" into MSCRIPT, instead of appearing on the video screen.

```
DEVICE | MSCRIPT
```

Advanced topics

Although you cannot redirect the standard input or output of a DO command (ZSHELL restriction), you may use redirection from within a Job Control Language JCL file. If standard input is redirected in a "command line" from within a JCL file, then ZSHELL will cause standard input to be retrieved from the specified file or device and NOT from the JCL file. This retrieval will continue until the particular application execution has terminated or until end-of-file is reached. If the standard input redirection is accomplished using the "<" specification, then the JCL will continue where it left off upon reaching end-of-file. If the standard input redirection is accomplished using the "<#" specification, JCL may or may not ABORT on reaching end-of-file depending on where within the execution the <BREAK> generated by ZSHELL was detected. If the "<@" option is used, then the JCL will be terminated when the standard input reaches end-of-file.

A few words are necessary concerning the use of the pound sign, "#". JCL already uses the "#" to indicate a substitution field when using compiled JCL. Therefore, if you are going to compile your JCL file, you must enter two "#" in a row to let JCL know you indeed meant to enter a "#". This means that you specify "<##INPUT/TXT". If, on the other hand, you are going to execute your JCL file without compilation (i.e. DO =) then use only a single "#".

It is very simple to write programs that utilize the abilities of ZSHELL. All input should be retrieved from standard input (*Kl) and all output should be sent to the

standard output. In BASIC, this would be "INPUT" and "PRINT" statements. In assembly language, you would use @KEY, @KEYIN, @DSP, and @DSPLY calls. The program may then be used with ZSHELL to allow you to redirect the input and the output. One version of a program may then access any device or file for input or output without any modifications.

For those of you who own MC, the C language compiler available from MISOSYS, specify the #OPTION REDIRECT OFF in your MC source program. Since ZSHELL will handle all command line I/O redirection, the code output by the compiler to handle redirection would never be used and only would serve to lengthen the object program unnecessarily.

When using the redirection of standard input, one should be aware that this feature may not always work as expected because of certain programming techniques. Take for an example a program that checks for a <BREAK> by constantly calling the @KBD vector and checking for X'80'. The program will be retrieving keystrokes which may have been meant for later responses. To deal with this problem, your programs should check for the <BREAK> key by checking the status of the KFLAG$.

To perform the piping functions, ZSHELL uses two files called Z0/PIP and Z1/PIP. These files are created by ZSHELL when you have entered PIPING specifications and are used during the execution of the programs being piped. They will no longer be used by ZSHELL once the piping is completed and may be killed later if you don't wish to have them cluttering up your disk. Under no circumstances kill either one of these files while piping is active.

Error messages explained

Approximately half of the resident portion of ZSHELL is taken up by file buffers and control blocks. Input and output each take a 256-byte buffer and a 32-byte File Control Block. Due to these memory requirements, the error messages have been kept short to avoid wasteful use of high memory. Because of this, there are only two error messages. They are explained below along with examples.

Redirection error

There are four major causes for this error message.

1. An attempt was made to redirect standard input or standard output more than once on a command. ZSHELL could redirect the standard output to a file or a device but NOT to both at the same time.

```
BASIC >*PR >TEST/TXT
```

2. An attempt was made to redirect standard input or standard output on a .-DO command. Redirection is possible within a JCL but NOT on the command that initiates that JCL.

```
DO JOB/JCL <INPUT/TXT
```

3. An attempt was made to redirect standard input to be retrieved from the *KI device. Standard input normally comes from the *KI.

```
MSCRIPT <*KI
```

4. An attempt was made to redirect standard output to the standard output device. When ">" is used, you cannot redirect the standard output to the *DO and when ">+" is used, you cannot redirect the standard output to the *PR.

```
SCRIPT >*DO
```

Piping error

There are three major causes of this error message.

1. An attempt was made to redirect standard input while the standard input was already being piped from a previous command.

```
DIR :0 | MSCRIPT <INPUT/TXT
```

2. An attempt was made to redirect standard output while the standard output was being piped to a following command.

```
DIR :0 >OUTPUT/TXT | MSCRIPT
```

3. An attempt was made to jump to the system vector @CMNDI while piping was active. If you want to execute a "/CMD" program from within an executing program, then use the @RUN system vector instead of @CMNDI. The @CMNDI vector is needed by ZSHELL to supervise the redirection function.

Parameter error

This is not an error message displayed by ZSHELL. It is possible to get this error from your application if you inadvertently forget to close your parameter string entries with the right parenthesis when you follow the parameters with redirection specifications.

Glossary of Terms

ABORT	Terminate an operation, usually with the <BREAK> key
ADDRESS	The location in memory of a particular byte, word, or string.
ALPHABETIC	Any of the ASCII lower case characters "a" through "z" or upper case "A" through "Z".
ALPHABETICAL	Pertaining to sorting a list of items in an order which relates to the items as alphabetic characters.
ALPHANUMERICS	Any of the ASCII characters including "alphabetic" and numeric, "0" through "9".
ARROW	A general term used to specify any of the four keyboard keys: <LEFT ARROW>, <RIGHT ARROW>, <UP ARROW>, and <DOWN ARROW>.
ASCENDING	A sorted order indicating an arrangement of items from lowest value to highest value. Sorted alphabetically, the arrangement would proceed from "AAAAA" through "ZZZZZ".
ASCII	An acronym for the American Standard Code for Information Interchange. This standard describes the character set ranging from a value of 0 (decimal) through 127 (decimal).
ATTRIBUTE	A facet of a file indicative of such things as whether the file is invisible to the DIRectory command, whether it's a SYSTEM file, whether it's a Partitioned Data Set, etc.
AUTO	A command provided in your operating system to schedule the automatic operation of a command line when your computer is started or restarted.

BACKSPACE	The keystroke used to back up and erase the character you previously typed. A standard Model 4 keyboard uses the <LEFT ARROW> key for backspace; model 4D computers have a <BKSP> key which duplicates the <LEFT ARROW>.
BACKUP	An operating system command used to make a duplicate copy of a diskette or group of files.
BANK	A term which refers to a portion of the extended memory of your computer. A "bank" represents 32,768 (32K) characters of storage.
BAS	A file extension used to designate a program written in the BASIC language.
BASIC	The computer language supplied with your disk operating system.
B1NARY	A number system consisting entirely of 0's and 1's; an environment which can exist in either of two states (usually ON or OFF).
BIT	The smallest unit contained in a byte which is composed of eight bits.
BOOT	The action of starting up your computer by turning on its power, inserting a system disk into the floppy drive numbered 0, closing the drive door, and possibly depressing the RESET switch.
BREAK	A key labeled as such on your keyboard; this key is used by PRO-WAM applications to either abort a command operation or exit from the application.
BUFFER	A specific memory space reserved usually for intermediate storage of data being written to or read from a disk sector.

Glossary of Terms

BUG	A malfunction of a computer program caused by an error in the program code. We hope there are none of these critters in this package.
BYTE	The smallest storage unit in your computer. byte can contain a number value of from zero through 255; a character is usually represented by a byte value. A byte contains eight bits.
CAT	A library command provided in TRSDOS 6.2 which performs the equivalent of "DIR (A=N)".
CHARACTER	A term applied to the symbols which make up the language we use to communicate with our computer and with which it communicates to us. These symbols may be alphanumerics, special characters (!@#$%^&*()-=+...), or non-printing control characters.
CIM	A file extension of files usually containing a Core Image Module. Such files are usually executable machine code in memory image form.
CLEAR	A key on your keyboard used to provide alternate functions when pressed in combination with another key or keys.
CMD	A file extension of executable command files. These are programs which can be run by typing the filename.
COM	A driver program provided with your operating system used to "connect" the DOS to your hardware serial port (RS-232C).
COMM	A utility program provided with your DOS to perform communications with another computer.
COMMAND	The general term applied to the many functions available in each application. The list of commands supported by a given application is usually displayed

as a menu in the bottom portion of the application window.

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

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

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

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

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

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

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

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

DAT	A file extension of files usually containing data.
DATA	Information usually stored in a disk file.
DCB	An acronym for Device Control Block; it is a system memory storage space associated with character devices (video, keyboard, printer, serial port, etc.).
DDEN	An acronym for double density formatting asso ciated with floppy diskettes.
DECIMAL	A number system based on the ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.
DEFAULT	A particular operation or configuration which is installed or designated without any specific action by you.
DELETE	The action of removing, killing, or erasing a file, a record, a field, or any similar entity.
DENSITY	A term applicable to the method of recording data onto a disk drive media. For floppy diskettes, the methods are usually single or double density.
DEPRESS	The action of tapping, momentarily pressing on, or otherwise engaging one of the keys on a keyboard.
DEVICE	A term applied to a peripheral component of your computer usually associated with character input/output (e.g. keyboard device, video device, printer device, etc.).
DEVSPEC	An abbreviation for "device specification". It represents an asterisk followed by the two-character name which identifies a device to the DOS.

DIGIT	One of the characters of a numbering system. Binary digits are the two characters 0 and 1; octal digits are the eight characters 0, 1, 2, 3. 4, 5, 6, and 7; decimal digits are the ten characters 0, 1. 2, 3, 4, 5, 6, 7, 8, and 9; and hexadecimal digits are the sixteen characters 0, 1. 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.
DIR	An operating system library command used to display the names and other storage details of the files stored on a floppy or hard disk.
DIRECTORY	A special file on a floppy or hard disk which contains the location and other pertinent details of all files stored on that disk.
DISK	The abbreviated term for "disk drive" which is the hardware apparatus used to either store your files, in the case of a hard or rigid disk drive, or is used to read your floppy diskettes, in the case of a floppy disk drive.
DISKETTE	A shorthand way of saying "floppy diskette", the flexible recording medium used by your computer's disk drives to store files.
DISPLAY	A shorthand way of referring to the video display screen of your computer.
DNARW	A mnemonic designating the <DOWN ARROW> key on your keyboard.
DO	An operating system command used to initiate the commands stored in a Job Control Language file (JCL).
DOS	An acronym for Disk Operating System. 'This is the system supplied with your computer which manages the files stored on diskettes and provides an operating environment for programs.
DRIVE	An abbreviation for "disk drive".

DRIVER	A special computer program connecting a hardware device to the DOS (e.g. COM/DVR).
DRIVESPEC	The disk drive identifier field of the file specification. This is indicated by a colon followed by a number in the range 0-7.
EOF	An abbreviation for "end of file". It is a position in a file indicating where the last byte is stored.
EXECUTE	An action which causes a computer program to start operating.
EXTENSION	One of the fields of a file specification generally used to indicate a particular class of file such as "/DAT" for data, "/CMD" for command, "/BAS" for BASIC, etc.
FIELD	One particular item in a group of data making up a record.
FILE	A collection of data records stored on disk.
FILENAME	The name field of a file specification.
FILESPEC	An abbreviation for file specification: the entire character string which identifies a particular file. It is composed of a file name, a file extension, a password, and a drivespec.
FILTER	A special program inserted logically into a device stream for the purpose of altering the behavior of input/output.
FLAG	A data field used to store a particular environment state. Flags are usually binary (e.g. ON/OFF, YES/NO, TRUE/FALSE).
FLOPPY	An abbreviation for "floppy diskette"; it is the flexible recording medium used to store files.

FLT	A file extension usually reserved for filter modules.
FORM	A pre-determined layout of fields positioned on the video display screen or on paper.
FORMAT	An operating system utility used to prepare a blank diskette before files can be stored on it, or other diskettes backed up to it.
FULL	One of the types of access provided to files. Others are READ, WRITE, EXEC, etc.
HEADER	Generally used to indicate a specific record attached to the front of a file which contains data used by the program or routine which reads the file.
HEXADECIMAL	A base-16 character value; hexadecimal digits are the sixteen characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B. C, D, E, and F.
HIT	An acronym for Hash Index Table. It is a portion of the directory stored on a disk which contains the one-byte directory entry codes for each active file.
HSB	An acronym for "high-order significant byte"; this is the highest storage portion of a three-byte value.
INSERT	The action of adding a character into a screen position by shifting all characters from the cursor position one place to the right.
INTEGER	A whole number, sometimes referred to as a counting number. Integer numbers have no frac tional part.
INVERSE	Another term for "reverse" associated with the videodisplay screen. Normal displayed characters are white foreground on a black background. Inverse, or reverse, video characters are black foreground on a white background.

INVISIBLE	An attribute which can be placed on a file by the ATTRIB DOS library command which inhibits the file's information from being presented by the DIR DOS library command.
INVOKE	The name applied to the function of gaining run access to a PRO-WAM application.
ISAM	An acronym for "Indexed Sequential Access Method"; used for a particular type of file which contains a sequential set of records or members with an index to the beginning of each record or member.
JCL	An acronym for Job Control Language. This is a DOS facility to execute a predetermined set of command lines.
KEY	One of the keys on your keyboard; sometimes used to refer to the computer action of scanning the keyboard to see if a key is depressed.
KEYIN	The acronym for "keyboard line input"; a DOS function used to obtain a line of keystrokes.
KEYSTROKE	The result of depressing one or more keyboard keys to generate a particular character value.
KEYWORDS	Character strings which represent a high level of importance when used in their environment.
KI	The name associated with the keyboard input device; designated as "*KI" when entering a device specification.
KSM	An acronym for KeyStroke Multiplication; a filter provided with your DOS which lets you generate a string of characters by depressing a single keystroke.

LIBRARY	A set of application modules stored together in one file and which contains an internal directory. Also the general term of many DOS commands identified by issuing a LIB command.
LOAD	The act performed by a LOADER.
LOADER	A program or routine used to read another file, which usually contains a computer program, and store it in memory to be run.
LOGICAL	The term applied to an event which is simulated rather than physical. A character value of 127 decimal may be interpreted as a carriage return for export purposes; since 127 is not physically a 13 decimal, it is termed, in this case, a logical carriage return.
LOWERCASE	Any of the alphabetic characters "a" through "z" entered normally by not depressing the <SHIFT> key nor by engaging <CAPS>.
LRL	An acronym for Logical Record Length; it represents the length of a record in bytes. Since the record length is not a physical phenomena. it is termed logical.
LSB	An acronym for "low-order significant byte"; this is the lowest storage portion of a three-byte value.
LS-DOS	A nomenclature of the 6.3 release of the operating system used on the Model 4 computer.
MACRO	The term applied to string of characters automatically generated from a single character. This is similar in theory to KSM.
MASK	A character or characters overlaying another character or characters in a special manner as to eliminate certain bit positions of the result.

MEDIA	A term indicating the magnetic or optical surface of material used to store computer files.
MEMBER	One of the applications stored in a library.
MEMBERSPEC	The specification of a library member which denotes its file name, its application name, and the disk drive on which the library file is stored.
MEMORY	The storage area for characters and programs internal to your machine (in contrast to disk storage).
MENU	The list of commands supported by an application which appear at the bottom of the application's window.
MESSAGE	A phrase or sentence used to inform you of some event. It may apprise you of an error or prompt you for a particular response.
MNEMONIC	An abbreviation or label for an entity.
MODEM	A hardware peripheral device used to communicate with another computer over a telephone or data line.
MODULE	Another name meaning "member".
MSB	An acronym for "mid-order significant byte"; this is the middle storage portion of a three-byte value.
MSPEC	An abbreviation for "memberspec".
NREC	An acronym for "number of records".
OCTAL	A base 8 numbering system; octal digits are the eight characters 0, 1, 2, 3, 4, 5, 6, and 7.
OFFSET	A position of a field, byte, or other piece of data which is relative to the beginning of a record.

ORIGIN	The memory address used to store the first byte of a program.
OVERLAY	A region of the DOS used for swapping in different processes.
OVERSTRIKE	The action of typing one character over another; the second replacing the first.
PaDS	An acronym for Partitioned Data Set. It is a name applied to the files managed by the utility of the same name; these files are similar to the DOS library files.
PARAMETER	An option, usually entered on the command line, which alters the behavior of a program.
PARTITION	Any one piece of a unit, commonly a hard disk drive, which has been divided up into more than one logical unit.
PARTSPEC	An incomplete file specification which is used to represent a class of files whose complete specification matches up with the partspec.
PASSWORD	A string of characters required as part of a file specification or entire disk before a given level of access is permitted.
PATCH	A DOS utility which applies alterations to disk files. The term also applies to the modifications.
PDS	Another "older" way of writing "PaDS".
PIPE	A connection, either logical or physical, between two programs created for the purpose of passing information from one to another.
PORT	A physical interface of your computer used to connect a peripheral piece of equipment (e.g. printer port, RS-232 serial port).

PR	The two character string following an asterisk which designate the printer device to the DOS.
PROGRAM	A predetermined sequence of machine instructions which together support some reasonably complex operations.
PROMPT	A displayed message which expects a usable response.
PROTECTION	Measures applied to a file for security purposes. It may relate to access level restrictions or password control before granting access. Protection is applied via the DOS ATTRIB command.
PRO-WAM	The trademarked name applied to the window controller and application manager published by MISOSYS.
PURGE	The act of erasing, deleting, removing, or killing a group of files.
RAM	A type of memory storage which provides both read and write capabilities.
README/TXT	A plain text file included on your program diskette which contains last minute information not printed in this manual.
RECORD	A collection of data fields associated with a particular key field or fields.
REGISTER	A very fast memory storage location located within the Central Processing Unit (CPU) of your computer.
REMOVE	The act of erasing, killing, or deleting a file, a module, a record, or other similar entity.
RENAME	The act of changing the filename and/or file extension of a file from one name to another.

RESIDENT	The term applied to a program which stays in the memory region of your computer after control is passed back to the DOS. The memory resident module usually embellishes, and is an extension to, the DOS.
RESPONSE	The entry you input into a program after viewing a particular message prompt.
RESTORE	The act of recovering a file archived to some place other than its working environment.
RETURN	Short for "carriage return"; see "CR".
REVERSE	Another term for "inverse" associated with the video display screen. Normal displayed characters are while foreground on a black background. Reverse, or inverse, video characters are black foreground on a white background.
RPN	An acronym for Reverse Polish Notation, commonly referred to as "infix". It relates to the syntax of entering numbers and operators for mathematical calculations.
RUN	The action of initiating program execution.
SCREEN	Short for the video display screen.
SCROLL	The action of shifting the entire text display of the screen or a window of the screen either vertically upward or downward.
SEARCH	The action of looking throughout a set of data records to find one which matches a key string.
SECTOR	A physical storage unit of a floppy diskette or hard disk drive.

SERIAL	A type of device where the character bytes are passed bit-wise; this contrasts with parallel where character bytes are passed as a full byte.
SETCOM	A library command provided with your DOS used to alter the parameters associated with the serial driver (COM/DVR).
SHELL	A program which performs some supervision over other programs and which provides the support for command processing.
SORT	The operation of placing a list of items into some designated order.
SPEC	Short for "file specification".
STRING	A series of character values, usually denoted on paper by enclosing in quotation marks (e.g. "this is a string").
STRUCTURES	Another form of record.
SVC	An acronym for "SuperVisor Call"; a facility of the DOS used to communicate with it at the program level in contrast to the command level.
SYNTAX	A particular requirement for the entering of information.
SYS	A particular file extension indicating a "system" file; e.g. SYS0/SYS.
SYSGEN	A command of the DOS used to store the current operating environment to the CONFIG/SYS file.
SYSTEM	The nomenclature of a diskette which is used to either BOOT your computer or which is used in drive :0.

TED	A text editor application included with the Mister ED application pack; also bundled with LS-DOS 6.3 as a stand alone command.
TEMPLATE	An outline or "form" used to organize a set of data.
TOGGLE	Switch a binary device from its current state to its opposite state.
TRACK	A circular virtual groove of a disk surface; it contains the sectors read during one rotation of the disk.
TXT	A file extension used to designate a plain text file.
UPARW	A mnemonic for the <UP ARROW> key.
USER	Someone who operates computer programs but does no or little computer programming.
UTILITY	A computer program designed to do some maintenance operation.
VIDEO	Short for the video display screen.
WILDCARD	A string of characters which don't completely designate a file specification but which contain certain "global" characters such as "*", "$", or "?" used to match any character thereby specifying a group of "matching" file specifications.
WORD	In the mathematical sense, the term applied to a 16-bit value which will occupy a two-byte storage region. Also is the name of the word processor used to prepare this documentation.