Flex™ Utilities

Technical Systems Consultants, Inc.
P.O. Box 2574
West Lafayette, Indiana 47906

FILEPAT UTILITY 26/05/80

```
**************************************************************************
*                              W A R N I N G                             *
*                                                                        *
* THIS UTILITY IS MEANT TO BE USED BY EXPERIENCED PROGRAMMERS ONLY       *
* WHO KNOW THE LAYOUT AND DESCRIPTION OF A DISK FILE AND RECORD.         *
* PATCHING THE WRONG BYTE OF DATA OR FLEX CONTROL INFORMATION  CAN       *
* LEAD TO THE DESTRUCTION OF THE PROGRAM BEING PATCHED. REFER TO THE     *
* " FLEX PROGRAMMER'S MANUAL ", PAGES 44 AND 45 FOR THE DESCRIPTIONS     *
* OF DATA SECTORS ON DISK, AS WELL AS DESCRIPTION OF A BINARY FILE.      *
*  THIS PROGRAM WILL PATCH THE FILE EVEN IF IT IS WRITE PROTECTED.       *
**************************************************************************
```

THE "FILEPAT" COMMAND ALLOWS PATCHING OF ANY TYPE OF FILE ON A FLEX 9.0 DISKETTE, EVEN THE BOOTSTRAP LOADER AND THE DISKETTE DIRECTORY.

DESCRIPTION

THE GENERAL SYNTAX OF THE COMMAND IS:

FILEPAT,<FILESPEC> OR FILEPAT

WHERE <FILESPEC> IS THE NAME OF THE FILE TO BE PATCHED.

ONCE THE COMMAND HAS BEEN ENTERED,THE PROGRAM WILL PROMPT YOU FOR A RELATIVE SECTOR NUMBER. THE RELATIVE SECTOR NUMBER IS A HEXADECIMAL VALUE FROM 00 TO FF, WHICH TELL THE PROGRAM WHICH SECTOR IN THE REQUESTED FILE WILL BE PATCHED, REGARDLESS OF WHERE ON THE DISK IT IS LOCATED. TO DETERMINE THE RELATIVE SECTOR NUMBER,USE THE 'DUMP' UTILITY TO FIND THE DATA TO BE PATCHED, AND THEN COUNT THE SECTORS FROM THE BEGINNING OF THE FILE, IN HEX, STARTING WITH SECTOR 00, UNTIL YOU REACH THE SECTOR CONTAINING THE DATA TO BE PATCHED. THIS WILL THEN BE THE RSN TO BE USED IN THE COMMAND.

ENTER FUNCTION C,D,P,R,V,S OR ?

THE USER CAN THEN USE ANY OF THE FOLLOWING FUNCTIONS TO MAKE PATCHES TO THE DATA READ FROM THAT SECTOR:

D= DISPLAY DATA
P= DISPLAY COMPLETE SECTOR
R= REPLACE DATA IN BUFFER
V= VERIFY IF EXISTING DATA IS CORRECT
C= CANCEL PROGRAM NO DATA IS WRITTEN TO DISK.
S= SAVE NEW DATA TO DISK
?= HELP DISPLAYS PROGRAM OPTIONS

THE DESIRED LETTER, PLUS ANY REQUIRED PARAMETERS ARE THEN ENTERED, FOLLOWED BY A CR . ONLY ONE FUNCTION PER LINE IS PERMITTED, SINCE THE

FLEX LINE BUFFER IS BEING USED, AND IT IS CLEARED AT THE END OF EACH FUNCTION EXECUTED. AFTER THE D,P,R OR V FUNCTIONS, THE PROGRAM WILL GO BACK TO THE INITIAL PROMPT "ENTER FUNCTION" . AFTER THE S OR C FUNCTION, A MESSAGE WILL BE OUTPUT ASKING YOU IF YOU WANT TO PATCH ANOTHER SECTOR. ANSWER 'Y' OR 'N' DEPENDING IF 'YES' OR 'NO'. IF YOU ANSWER 'N', THEN THE PROGRAM TERMINATES AND RETURNS TO FLEX.

THE FOLLOWING EXAMPLES OF ALL FUNCTIONS REFER TO THE FILE ASMB.CMD FILE, SINCE ALL USERS OF FLEX 9.0 HAVE IT, BECAUSE IT WAS SUPPLIED BY TSC AS PART OF FLEX 9.0.
BEFORE ATTEMPTING ANY PATCHES FOR THE PURPOSE OF LEARNING HOW TO USE THIS PROGRAM, MAKE SURE THAT YOU HAVE A SPARE COPY OF ASMB.CMD ON A SEPARATE DISK, IN CASE YOU INADVERTENTLY PATCH SOME WRONG DATA IN THE PROGRAM, AND IT NO LONGER WORKS. ALL EXAMPLES GIVEN REFER TO ASMB.CMD.0 'RSN' 0F.

FILEPAT,ASMB.CMD.0

D(ISPLAY) FUNCTION

SYNTAX:

D(,<ADDR>)(,<BTCT>)

WHERE D= DISPLAY FUNCTION
ADDR = ADDRESS OF DATA WITHIN THE SECTOR, WITHIN A RANGE
OF 00 TO FF IN HEX. DEFAULT VALUE IS 00.
BTCT = BYTE COUNT, NUMBER OF BYTES, IN HEX, TO BE DISPLAYED.
THE DEFAULT VALUE IS 16 (HEX 10).

EXAMPLE 1:

D (CR) (DEFAULTS TO ADDRESS=00,BYTECOUNT=HEX 10)

00 0F 0A 00 10 AC 11 DE 45 FF AC 14 CE A8 40 86 01 .....E.....@..

THIS WILL DISPLAY 16 BYTES OF DATA FOLLOWED BY THE ASCII EQUIVALENTS. IF THERE IS NO ASCII EQUIVALENT, A PERIOD (.) IS PRINTED. IF YOU TRY THIS ON YOUR SYSTEM WITH THE ASMB.CMD.0 FILE YOU WILL FIND THAT THE FIRST 2 BYTES ARE DIFFERENT, SINCE THESE REFER TO THE TRACK AND SECTOR OF THE NEXT SECTOR IN THE FILE, AND PROBABLY THEY ARE NOT THE SAME AS MINE. THE REST OF THE DATA SHOULD BE IDENTICAL.

EXAMPLE 2:

D,3C,2

3C B3 97 ..

THIS WILL DISPLAY 2 BYTES OF DATA STARTING AT ADDRESS HEX 3C IN THE SECTOR.

EXAMPLE 3:

```
D,42,37

42      Ø6 BD Ø6 99 5A 26 FA CE Ø0 D1 BD Ø6 8A BD Ø5 27    ....Z&.........'
52      96 47 26 Ø8 CE 11 5D BD Ø6 8A 20 33 7F Ø0 45 B6    .G&... ... 3 .E.
62      AC ØF 81 Ø9 22 Ø3 BD Ø2 Ø6 B6 AC Ø2 10 F1 FF ØE    ....".............
72      81 Ø9 22 Ø6 BD Ø2 Ø6    .."....
```

ANY ATTEMPT TO DISPLAY DATA PAST THE SECTOR LIMIT OF FF WILL RESULT IN TRUNCATION OF THE DATA DISPLAYED TO KEEP IT WITHIN THE DESIRED SECTOR.

IT SHOULD BE REMEMBERED THAT THE DATA IS BEING DISPLAYED FROM THE SECTOR BUFFER IN THE FCB AND NOT DIRECTLY FROM DISK. AFTER CHANGES ARE MADE TO THE SECTOR BUFFER, THE FINAL COPY WILL BE WRITTEN BACK TO DISK WITH THE S(AVE) FUNCTION.

P(RINT) FUNCTION

SYNTAX:

P

NO ADDITIONAL PARAMETERS ARE REQUIRED. IT WILL DISPLAY THE CURRENT CONTENTS OF THE SECTOR BUFFER IN 16 LINES OF 16 BYTES PER LINE, PLUS THE ASCII TRANSLATABLE DATA. THIS IS USEFUL FOR FINDING THE DATA TO BE PATCHED, AS WELL AS CHECKING THE DATA, AFTER THE PATCH IS DONE, BUT BEFORE IT IS WRITTEN BACK TO DISK, TO MAKE SURE THE PATCH WAS DONE CORRECTLY.

R(EPLACE) FUNCTION

SYNTAX:

R,<ADDR>,<DATA1>(,<DATA2>)(,<DATA3>)(,<DATAN>) ETC.

WHERE R = THE REPLACE DATA FUNCTION
ADDR = ADDRESS, IN HEX OF THE START ADDRESS OF THE BYTES TO
BE REPLACED, WITHIN THE RANGE OF Ø0 TO FF, AND
HAVING A DEFAULT VALUE OF Ø0
DATA1 = ONE BYTE OF HEX DATA TO BE PATCHED.
EACH BYTE MUST BE 2 CHARACTERS IN THE RANGE OF Ø TO 9 OR A TO F. EACH ADDITIONAL BYTE MUST BE SEPARATED FROM THE PREVIOUS ONE BY A SPACE OR COMMA (,). THE MAXIMUM NUMBER OF BYTES THAT CAN BE REPLACED IS LIMITED TO 40 BYTES PER LINE.

AFTER THE DATA IS ENTERED,FOLLOWED BY A CR, THE PROGRAM WILL OUTPUT THE START ADDRESS AND THE OLD DATA THAT IS BEING REPLACED.AT THIS TIME IT WILL ASK IF THAT IS WHAT YOU WANT TO REPLACE, WITH THE QUESTION,

"VERIFY OK?"

IF YOU ARE SURE YOU WANT TO REPLACE THE OLD DATA, ANSWER "Y" AND A CR.

AT THIS POINT THE PROGRAM WILL OUTPUT THE NEW DATA THAT HAS BEEN PUT IN THE SECTOR BUFFER, AND GO BACK TO THE MAIN PROMPT:

ENTER FUNCTION C,D,P,R,V,S OR ?

IF YOU HAVE DECIDED THAT YOU DID NOT WANT TO MAKE THE CHANGE, WHEN THE PROMPT "VERIFY OK?" IS OUTPUT, YOU CAN ANSWER WITH AN "N" AND A CR, OR SIMPLY A CR. AND NO DATA WILL BE CHANGED IN THE SECTOR BUFFER.

YOU MAY MAKE MULTIPLE PATCHES TO THE SAME SECTOR. ALL CHANGES ARE MADE IN THE SECTOR BUFFER ONLY AND NOT ON THE DISK UNTIL YOU USE THE "S" FUNCTION TO RE-WRITE THE CONTENTS OF THE SECTOR BUFFER BACK TO DISK.

V(ERIFY) FUNCTION

SYNTAX:

V,<ADDR>,<DATA1>(,<DATA2>)(,<DATA3>) ETC.

WHERE V= VERIFY FUNCTION
ADDR = ADDRESS OF FIRST BYTE TO BE CHECKED
DATA1 = DATA YOU EXPECT TO BE AT THE ADDRESS YOU SPECIFIED.ANY FOLLOWING BYTES MUST BE SEPARATED BY A SPACE OR A COMMA (,).

IF THE DATA EXACTLY MATCHES, THE PROGRAM WILL OUTPUT "DATA VERIFIES OK", AND WILL RETURN TO THE MAIN PROMPT.

IF THE DATA DOES NOT MATCH, THE PROGRAM WILL OUTPUT "DATA DOES NOT VERIFY", AND THE PROGRAM WILL TERMINATE.

THE PRINCIPAL USE OF THIS FUNCTION IS FOR USE WITH AN INPUT FILE CONTAINING PATCH DATA. IN THIS CASE, YOU WOULD ISSUE THE VERIFY FUNCTION BEFORE THE R FUNCTION, TO MAKE SURE THAT THE CORRECT DATA IS BEING REPLACED, AND IF IT VERIFIES OK, THE REPLACE WILL BE DONE. IF IT DOES NOT VERIFY, SOMEHOW THE DATA THAT YOU EXPECTED TO BE THERE IS NOT THERE(YOU MAY BE AT THE WRONG SECTOR OR WRONG ADDRESS, OR A PATCH HAS ALREADY BEEN MADE THERE) AND THE PATCH IS ABORTED AT THIS TIME, TO PREVENT PATCHING OF THE WRONG DATA.

EXAMPLE 1:

V,80,2D WILL VERIFY THAT THE BYTE AT ADDRESS HEX 80 IS A '2D'

R,80,2F WILL CHANGE THE BYTE AT ADDRESS HEX 80 FROM '2D' TO '2F'.

IF THE BYTE AT HEX 80 HAD NOT BEEN A '2D' THE PATCH WOULD NOT BE MADE, AND THE PROGRAM WOULD RETURN TO FLEX.

S(AVE) FUNCTION

SYNTAX:

S

WHERE S WILL SAVE THE CONTENTS OF THE SECTOR BUFFER TO DISKETTE, OVERLAYING THE ORIGINAL SECTOR, AND VERIFYING THAT IT HAS BEEN WRITTEN BACK CORRECTLY;(NO DISK WRITE ERRORS).

A MESSAGE WILL BE OUTPUT GIVING THE STATUS OF THS SAVE OPERATION, AND THE PROGRAM WILL ASK YOU IF YOU IF YOU WANT TO PATCH ANOTHER SECTOR.

C(ANCEL) FUNCTION

SYNTAX:

C

WHERE C WILL CANCEL THE PROGRAM WITHOUT WRITING ANYTHING TO DISK, ISSUE AN ABORT MESSAGE, AND PROMPT YOU FOR ANOTHER SECTOR TO BE PATCHED.

? HELP FUNCTION

THIS FUNCTION WILL PRINT OUT ON YOUR TERMINAL A BRIEF DESCRIPTION AND SYNTAX OF ALL OF THE ABOVE FUNCTIONS, IN CASE YOU FORGOT WHAT THE INDIVIDUAL LETTERS MEANT, AND YOU DON'T WANT TO LOOK THEM UP IN THE DOCUMENTATION.

EXAMPLES

THE FOLLOWING EXAMPLES ARE SUPPLIED FOR USING THE FILEPAT UTILITY, USING AN INPUT TEXT FILE FOR INPUT RATHER THAN FROM THE TERMINAL.

PATCH1.TXT CONTAINS DATA TO PATCH ASMB.CMD.0 FILE TO CHANGE THE DATE PRINTOUT FROM MM-DD-YY TO MM/DD/YY.

PATCH2.TXT WILL REVERSE THE PATCH1 PROCEDURE TO PUT IT BACK THE WAY IT WAS.

TO EXECUTE THESE PATCHES, ENTER THE FOLLOWING COMMAND:

(P,)I,PATCH1,FILEPAT,ASMB.CMD.Ø

THE (P,) IS OPTIONAL; IT WILL DIRECT ALL OUTPUT TO THE SYSTEM PRINTER, SO YOU COULD KEEP A PERMANENT RECORD OF THE PATCH. IF YOU WANT TO RUN PATCH2 TO CHANGE THE DATE FORMAT BACK FROM MM/DD/YY TO MM-DD-YY, THEN ENTER THE FOLLOWING COMMAND:

(P,)I,PATCH2,FILEPAT,ASMB.CMD.Ø

FILEPAT IS A .CMD FILE ON THE SYSTEM DRIVE.
ASMB.CMD IS A .CMD FILE ON THE SYSTEM DRIVE.
I IS A .CMD FILE ON THE SYSTEM DRIVE.
PATCH1 IS A .TXT FILE ON THE WORK DRIVE.
PATCH2 IS A .TXT FILE ON THE WORK DRIVE.

WHEN MAKING UP YOUR OWN PATCH FILES,YOU MAY USE THE 'BUILD' UTILITY, OR THE TEXT EDITOR. MAKE UP THE FILE EXACTLY AS YOU WOULD ENTER THE RESPONSES FROM A TERMINAL.

THE SEQUENCE SHOULD BE:

1. SECTOR NUMBER TO BE PATCHED.
2. VERIFY THE DATA TO BE PATCHED.
3. REPLACE THE DATA
4. Y (ANSWER TO VERIFY OK?)
5. HERE YOU MAY REPEAT 2,3,4 USING DIFFERENT DATA, AS LONG AS IN THE SAME SECTOR. YOU MAY ALSO USE THE D OR P FUNCTIONS IF YOU WANT A PRINTOUT OF THE RESULTS, TO SAVE FOR YOUR RECORDS OF PATCHES DONE.
6. S (SAVE CONTENTS OF SECTOR BUFFER TO DISK)
7. N (ANSWER TO 'DO YOU WANT TO PATCH ANOTHER SECTOR?') IF YOU WANT TO PATCH ANOTHER SECTOR AT THIS POINT, ANSWER 'Y' TO THE QUESTION, AND GO BACK TO ITEM 1 GIVING NEW SECTOR NUMBER, ETC. THE LAST ENTRY IN YOUR FILE MUST BE A 'N' TO TERMINATE THE PROGRAM AND RETURN TO FLEX.

NOTE: YOU MAY ADD YOUR OWN COMMENTS IN THE PATCH FILE AFTER THE COMMAND,FOR DOCUMENTATION PURPOSES. THE COMMAND SHOULD BE FOLLOWED BY A SPACE, A SEMI-COLON, AND THEN YOUR COMMENTS.

TAILORING TO YOUR SYSTEM.

THERE ARE TWO VARIABLES YOU MAY WANT TO CHANGE IN THE PROGRAM, DEPENDING ON YOUR PARTICULAR TERMINAL. ONE WILL LET YOU SET THE SIZE OF THE DISPLAY TO FIT EITHER A 64 CHARACTER CRT OR TERMINAL,OR YOU CAN SET THE SWITCH UP FOR USE ON 72 CHARACTER TERMINALS OR LARGER. THIS IS DONE SIMPLY BY EDITING THE SOURCE PROGRAM, AND CHANGING THE VALUE OF THE FCB AT LABEL 'CRTSW' AS FOLLOWS:

CRTSW FCB ØØ FOR TERMINALS OF LESS THAN 72 CHARACTERS.
CRTSW FCB 72 FOR ANY TERMINALS EQUAL OR GREATER THAN 72 CHARACTERS PER

LINE.

THE SECOND VARIABLE IS THE START ADDRESS OF THE PROGRAM WHICH NOW IS ORG'ED AT $0000. YOU MAY WANT IT AT SOME OTHER ADDRESS, SO JUST CHANGE THE 'ORG' STATEMENT AND RE-ASEMBLE THE PROGRAM. DO NOT ATTEMP TO 'ORG' IT AT $A100 SINCE IT WILL NOT FIT IN FLEX UTILITY COMMAND SPACE.

THIS IS AN ORIGINAL PROGRAM WRITTEN BY

MILAN KONECNY
193 CHAPLEAU AVE
DOLLARD-DES-ORMEAUX
P.Q. CANADA
H9G 1C3

FOR USE BY '68' MICRO JOURNAL MAGASINE. AN EARLIER VERSION HAS BEEN DISTRIBUTED BY THE 'FLEX USERS' GROUP NEWSLETTER ON A FREE BASIS .

THE AUTHOR WISHES TO HEAR FROM ANY USERS OF THIS PROGRAM AS TO SUGGESTIONS FOR IMPROVEMENT, OR ANY OTHER COMMENTS, AND WILL BE WILLING TO SUPPLY THE PROGRAM AS SOURCE CODE ON STANDARD 5.25 INCH FLEX 9.0 DISKETTES TO THOSE WHO SEND ME A BLANK DISKETTE PLUS $2.00 FOR POSTAGE

CONTENTS

=> { DELETE, FILE1
RENAME, FILE2, FILE1

use REPLACE, FILE1, FILE2,

3X

CONTENTS

FILED ALPHABETICALLY !

BASREF - A BASIC Cross Reference Program

This program is designed to provide the BASIC user with additional information, besides the listing of the program, to write or make changes to his files. The output of the program lists each line number in the BASIC program and each line number that referenced that line, possibly with a GOTO or GOSUB instruction. Additionally, all variables are printed, along with the line number that referenced them.

The systax is:

BASEREF <DRV.NU.><FILENAME> or
P BASEREF <DRV.NU.><FILENAME> for a hard-copy printout.

The default extension is .BAS.

CHECK

The CHECK utility is used to compare two disk files. The reult of the comparison will be reported to the terminal.

DESCRIPTION

The general syntax of the CHECK command is:

```
CHECK,<file spec 1>,<file spec 2>
```

where the file specs default to a TXT extension and to the working drive. File one will be read and compared against file two one character at a time. The files may be text or binary type files. The result of the comparison will be reported to the terminal (files are identical or not). An example follows:

```
+++CHECK,REPORT1,REPORT2
```

This command line would cause the file named REPORT1.TXT on the working drive to be compared to the file named REPORT2.TXT.

CMPMEM

The CMPMEM command compares the contents of a binary file on the disk to the contents of memory where it should be loaded. This is useful for program debugging and memory problem detection.

DESCRIPTION

The general syntax of the CMPMEM command is:

```
CMPMEM,<file spec>
```

where the file spec defaults to a BIN extension and to the working drive. The file specified will be read just as if it were to be loaded into memory, but instead, each byte will be compared to what already exists in memory. If any differences are found, they will be printed out as the address, followed by the data in memory at that location, followed by the data from the disk file. All differences will be printed on the output device. An example follows:

```
+++CMPMEM,FENCE
```

This would cause the file named FENCE.BIN on the working drive to be read and compared to the actual memory contents throughout the load address range of the file.

CONCAT

The CONCAT command allows the listing and concatenation of several files. The files will be listed to the output device (file if the O command is used), one after the other.

DESCRIPTION

The general syntax of the CONCAT command is:

CONCAT,<file spec list>

where the file spec list is a list of file specifications separated by commas. The file specs will default to a TXT extension and to the working drive. An example follows:

+++CONCAT,CHAPT1,CHAPT2,CHAPT3,CHAPT4

This will display the contents of the files CHAPT1.TXT, CHAPT2.TXT, CHAPT3.TXT, and CHAPT4.TXT, one after the other on the terminal.

CONTIN

The CONTIN command is intended for use in repeating or complex EXEC command files. It prompts the terminal for a YES or NO response for continuing the files execution.

DESCRIPTION

The general syntax of the CONTIN command is:

```
CONTIN
```

Executing CONTIN will cause the massage 'CONTINUE (Y-N)? ' to be displayed on the terminal.. A 'Y' response will cause the EXEC program to execute the next command in the command file. A 'N' response will cause FLEX to regain control and the EXEC program will be halted. This utility is useful for incorporating into EXEC command files which repeat themselves (by calling itself as the last line of the command file). The CONTIN command provides a mechanism for escape from this ever repeating type of command file.

DATE

The DATE utility allows the setting and displaying of the system date register. The date register is used by other FLEX utilities.

DESCRIPTION

The general syntax of the DATE command is:

```
DATE[,<date spec>]
```

where the date spec is in the form MM,DD,YY. If the date spec is left off the command line, the date will be displayed on the terminal. A few examples follow:

```
+++DATE
+++DATE,10,6,78
```

The first example would display the current date on the terminal. The second example would set the date to 'October 6, 1978'. One of the FLEX utilities which make use of the system date register is the Text Processor.

DECOMPIL

THE XBASIC DECOMPILER

Syntax: DECOMPIL <DR#>.<FILENAME>[<.EXT>]

The file extension will default to a '.BAC' extension if none is typed in. When the decompiler asks if you want to BUILD a disk file, type 'Y' if you do and 'N' if you don't. The decompiler will then prompt for a filename of the program that will be created. The DEFAULT EXTENSION for the new file will be a '.BAS' and DECOMPIL will exit back to FLEX if the file already exists, albeit non-destructively. (DECOMPIL WILL NOT ERASE ANY FILES!).

For a printout, type; P,DECOMPIL,FILENAME.EXT

DECOMPIL will follow all of the TTYSET parameters, so use them for a neater looking printout, as line length generally exceeds the normal 80 characters. DECOMPIL will, in some cases, put in extra spaces for a better looking printout. XBASIC will not accept over 127 characters and will give ERROR #51 (ILLEGAL CHARACTER IN LINE) when you try to load a long line. While in XBASIC, type "LIST" and the last line printed was the last line that made it in. You will have to EDIT the created file and change lines that are greater than 127 characters (remove spaces, make two lines out of one, etc).

NOTE... DECOMPIL will take up to 255 characters of .BAC, so it will decompile longer lines, such as created using the PRECOMPILER.

DECOMPIL will only decompile T.S.C. XBASIC (EXTENDED BASIC),and will not work for other BASIC compilers (including T.S.C. regular BASIC). Also, you should have at least 16K of work RAM.

DUMP

The DUMP utility is used for dumping the contents of a file, one sector at a time, in both hex and ASCII characters. It can be used as a disk debugging aid or to clarify the exact format for disk files.

DESCRIPTION

The general syntax of the DUMP command is:

    DUMP,<file spec>

where <file spec> specifies the file to be dumped and defaults to a BIN extension. As each sector is displayed it will be preceeded by two, 2 digit numbers, the first being the hex value of the track number, the second being the sector number of the sector being dumped. Each data line will contain 16 hex digits representing the data followed by the ASCII representations of the data. All non-printable characters are displayed as underscores (_). An example follows:

    +++DUMP,FILE55

This would cause the contents of each one of the sectors contained in the file named FILE55.BIN to be dumped on the output device.

DUP

The DUP command is used to list the file names contained in one disk's directory which are not duplicated in a second disk's directory. This is a useful utility for comparing files on original disks to those on a backup diskette.

DESCRIPTION

The general syntax of the DUP command is:

```
DUP,<drive number>,<drive number>
```

where the disk's directory in the first drive number specified is to be compared to the second. The files whixh exist on the first drive but not on the second will be listed on the output device. As an example:

```
+++DUP,0,1
```

This would cause all file names which were on the diskette in drive zero but not on drive one to be listed on the output device.

DIR

The DIR utility is similar to the CAT command but displays all directory information associated with the file. This command gives a detailed look at the disk directory.

DESCRIPTION

The general syntax of the DIR command is:

```
DIR[,<drive list>][,<match list>]
```

where <drive list> and <match list> are the same as described in the CAT command. Each file name will be listed with its file number, starting disk address in hex (track-sector), ending disk address, and file size in number of sectors. On the larger FLEX systems, the file creation date and attributes will also be displayed. At the end of the DIR list, a disk file use summary is printed, giving the total number of files, the number of sectors used by those files, the remaining number of sectors (free sectors), and the size of the largest file found on the disk. The 'file number' associated with a file represents that files location in the directory, so the file numbers may not be consecutive if a lot of files have been deleted from the disk. A few examples follow:

```
+++DIR
+++DIR,1,A.T,FR
```

The first example would list all files on the working drive. The second example would list only those files on drive 1 whose names began with 'A' and extensions began with 'T', as well as those files whose names started with 'FR'.

ECHO

The ECHO command is a very useful utility for incorporation into EXEC command files. It allows the echoing of ASCII strings to the terminal.

DESCRIPTION

The general syntax of the ECHO command is:

```
ECHO,<string>
```

where <string> is any string of printable characters terminated by a carriage return or end of line character. A few examples of the ECHO command follow:

```
+++ECHO,THE COPY PROCESS IS STARTING
+++ECHO,TERMINAL 12
```

The first example would print the string "THE COPY PROCESS IS STARTING" on the terminal. The second example would print "TERMINAL 12". It is often useful to use ECHO in long EXEC command files to send instructive messages to the terminal to inform the operator of the status of the EXEC operation.

FIND

The FIND command is used for finding all lines in a text file containing a specified string. It is faster to use FIND than to enter the editor to find strings.

DESCRIPTION

The general syntax of the FIND command is:

```
FIND,<file spec>,<string>
```

The file spec defaults to a TXT extension and to the working drive. The string may be any printable characters (non-control characters) and is terminated by the carriage return or end of line character. Upon execution, all lines containing the specified string will be printed on the terminal preceeded by that lines line number. When finished, the total number of lines found containing the string will be printed. Following are a few examples.

```
+++FIND,TEXT,THIS IS A TEST
+++FIND,BOOK.TXT,OHIO
```

The first example would find and display all lines in the file TEXT.TXT which contained the character string "THIS IS A TEST". The second example would search the file BOOK.TXT for the string "OHIO" and list all lines found.

FILTYP

The FILTYP command is used to determine the type of a file, either binary or text. This is useful when non-standard extensions have been used and the file type has been forgotten.

DESCRIPTION

The general syntax of the FILTYP command is:

```
FILTYP,<file spec>
```

where the file defaults to the working drive. Upon executing this command, the system will report the file to be either a TEXT type file, or a BINARY type file (a file which may be loaded into memory). An example will demonstrate its use.

```
+++FILTYP,MYSTERY.XYZ
```

The system will report the type of the file named MYSTERY.XYZ found on the working drive.

FREE

The FREE command is used to report the total number of free (available) sectors on a diskette. The approximate number of kilobytes remaining is also reported.

DESCRIPTION

The general syntax of the FREE command is:

```
FREE[,<drive number>]
```

If the drive number is not specified it will default to the working drive. An example follows:

```
+++FREE,1
```

This command line will report the number of available sectors and approximate number of kilobytes remaining on the disk in drive 1.

FILES

The FILES utility is similar to the CAT command but displays only the file names and extensions. This command is useful for getting a short and quick report of the directory contents.

DESCRIPTION

The general syntax of the FILES command is:

```
FILES[,<drive list>][,<match list>]
```

where <drive list> and <match list> are the same as described in the CAT command. The file names will be listed across the page and in a columnar fashion. The number displayed per line is determined by the TTYSET Width parameter. If the Width is zero, 80 columns are assumed to be available and 5 names will be listed on each line. Smaller Width values will result in fewer names per line being displayed. A few examples follow:

```
+++FILES
+++FILES,1,A.T,FR
```

The first example would list all files on the working drive. The second example would list only those files on drive 1 whose names began with 'A' and extensions began with 'T', as well as those files whose names started with 'FR'.

FLIST

The FLIST utility is used to get a page formatted listing of a text type file. It is similar in operation to the LIST utility.

DESCRIPTION

The general syntax of the FLIST command is:

```
FLIST,<file spec>[,<line range>][,+<options>]
```

where the file spec designates the file to be listed and defaults to a TXT extension and to the working drive. The <line range> is the same as described in the LIST utility. If no range is specified, all lines will be displayed. Two options are supported, 'N' for line numbers, and 'P' for pagination. If the P option is specified, FLIST will prompt for a title. The title may contain a maximum of 40 characters. Each page will then be listed with a title and page number, followed by 54 lines of text (numbered if the N option was specified), and a hex $0C formfeed character. A few examples will demonstrate the use of FLIST.

```
+++FLIST,CHAPTER1
+++FLIST,LETTER,10-100,+NP
+++FLIST,TEXT,50
```

The first example would cause the file named CHAPTER1.TXT to be displayed on the screen without line numbers or pagination. The second example would list LETTER.TXT from line number 10 through line 100 with line numbers and pagination. The last example would list the file named TEXT.TXT from line 50 to the end of the file. No line numbers will be output since the 'N' option was not specified.

HECHO

The HECHO command is used for sending special character strings to the terminal. It is similar to the ECHO command, but HECHO allows control characters as well.

DESCRIPTION

The general syntax of the HECHO command is:

```
HECHO,<hex string>
```

where <hex string> is a list of hex digits representing ASCII characters. A few examples will demonstrate the use of HECHO.

```
+++HECHO,C
+++HECHO,D,A,0,0,0,0
+++HECHO,7,54,45,53,54,7
```

The first example will output a page eject (hex C) to the terminal. The next example will output a carriage return (hex D), a line feed (hex A), and then 4 null characters (hex 0). The last example will will output an ASCII bell character (hex 7), then the string 'TEST', followed by another bell character.

INSTALL

The INSTALL utility is used as a convenient way of renaming a file with a .BIN extension to the same name but with a .CMD extension. In effect, you are 'installing' the file into the utility command set.

DESCRIPTION

The general syntax of the INSTALL command is:

```
INSTALL,<file spec>
```

where the file spec defaults to a BIN extension and to the working drive. This utility has the same affect as using 'RENAME,FILE.BIN,FILE.CMD'. An example will demonstrate its use.

```
+++INSTALL,LOAD
```

This command line would cause the file named LOAD.BIN on the working drive to be renamed LOAD.CMD. This command is simply a time saver for those who do not like to type names twice!

INTEG

The INTEG command is used to completely test the free space (unused sectors) on a diskette. This routine will guarantee the integrity of the available disk space.

DESCRIPTION

The general syntax of the INTEG command is:

```
INTEG[,<drive number>]
```

where the drive number specifies which disk is to be tested and defaults to the working drive. This program will check that the free space contains the correct number of sectors, that it starts at the correct disk address, and that it terminates at the correct disk address. If any discrepencies are found the appropriate error message will be displayed, otherwise, the message 'FREE SPACE ALL OK!' will be output. An example follows:

```
+++INTEG,0
```

This would test the free space on the disk in drive 0. It should be noted that INTEG may require a moderate amount of time to run. Any diskette not passing the INTEG test should not be used for creating new files. This command does not test any of the disk space being used by files on the disk.

LOW-UP

The LOW-UP command is used to convert a file into all upper case letters. It is useful for those systems unable to work with lower case letters.

DESCRIPTION

The general syntax of the LOW-UP command is:

LOW-UP,<input file spec>,<output file spec>

The input file spec specifies the name of the file needing the conversion, and the output file spec specifies the file name of the new converted file. Both default to a TXT extension and to the working drive. The new file will end up containing only upper case letters and the original file will be left unchanged. An example follows:

+++LOW-UP,LISTER,LISTERU

This would cause a file named LISTERU.TXT to be created which is identical to the file named LISTER.TXT except all letters will be upper case.

MAP

The MAP utility is used for determining the load addresses and transfer address of a binary file. This command is useful in conjunction with the SAVE command.

DESCRIPTION

The general syntax of the MAP command is:

```
MAP,<file spec>
```

where the file spec defaults to a BIN extension and to the working drive. The beginning and ending addresses of each block of object code will be printed on the terminal. If a transfer address is contained in the file, it will be printed at the end of the list of addresses. If more than one transfer address is found in a file, only the effective one (the last one encountered) will be displayed. An example will demonstrate the use of MAP.

```
+++MAP,MONITOR
```

This command line would cause the load addresses and transfer address (if one exists) of the file named MONITOR.BIN to be displayed at the terminal.

=> MAP.CMD,2)*BASIC.CMD.0

0220 - 0853
0873 - 47B2
0100 TRANSFER ADDRESS i.e. START OF PROGRAM.

MEMTEST

The MEMTEST command executes a memory pattern test. This test will detect 99% of all memory problems if allowed to run a sufficient amount of time.

DESCRIPTION

The general syntax of the MEMTEST command is:

MEMTEST,<hex start address>,<hex end addres>

The start address is a hex number stating where in memory the test program should start testing, and the end addres is the last location to be tested. The test fills memory with random numbers, goes back and checks the numbers are correct, and then repeats the process. This test should be allowed to run approximately 1 hour for each 4K block of memory being tested. Remember not to specify a memory range which will overlap the test program itself. Each successful pass through the test will be shown by the printing of a '!' on the terminal. An example follows:

+++MEMTEST,0,3FFF

This would test memory from location 0 through location 3FFF. The system RESET must be used to exit the test program. Reboot the system to use FLEX.

MEMDUMP

The MEMDUMP command allows the dumping or displaying of a selected portion of memory at the terminal in both hex and ASCII. It is very useful as a diagnostic aid.

DESCRIPTION

The general syntax of the MEMDUMP command is:

MEMDUMP[,<start address>]

where the start address is the hex value of the address at which dumping should start. If the address is left off of the command line, dumping will begin at address 0000. The display consists of 16 lines of data. Each line starts with the address in memory from which the data is taken. Following the address is 16 bytes of data displayed as 2-digit hex numbers. Finally, 16 ASCII characters are printed which represent the data bytes just printed. All control characters are printed as underscores (_). After each block of 256 bytes (16 lines of data), the program stops and expects an input character. Typing an 'F' will cause the display to move Forward, printing the next sequential 256 bytes. Typing a 'B' will move Backwards, printing the previous 256 byte block. A carriage return will return control to FLEX. All other characters are ignored. An example follows:

```
+++MEMDUMP,A00
```

This would cause memory to be dumped starting at location hex 0A00. The hex and ASCII representations are displayed.

MEMOVE

The MEMOVE command will move any block of memory to any other specified memory location.

DESCRIPTION

The general syntax of the MEMOVE command is:

```
MEMOVE,<start address>,<end address>,<destination>
```

where all addresses are specified in hex. The start and end addresses specify the bounds of the block of memory to be moved, and the destination address designates the location to where it should be moved. An example follows:

```
+++MEMOVE,400,4FF,1A00
```

This command line would cause the block of memory from location hex 400 through hex 4FF to be moved to location hex 1A00. The data which was from 400 through 4FF will now exist from 1A00 to 1AFF. The original block of data is left unchanged.

OLOAD

The OLOAD command is used to load a binary file into memory with a specified offset address. No code is modified during the load. This utility is useful for PROM programming applications.

DESCRIPTION

The general syntax of the OLOAD command is:

```
OLOAD,<file spec>[,<offset>]
```

where the file spec defaults to a BIN extension and to the working drive. The optional offset is a hex value which is to be added to the normal load address. If the offset is not specified, it is assumed zero which causes OLOAD to act exactly like the GET command. The offset addition will wrap around the end address of $FFFF. As an example, if a file normally loaded at location $6000, and an offset of $A000 was specified, the file would be loaded at $0000. An example of the OLOAD command follows:

```
+++OLOAD,XDATA,2000
```

This would cause the file named XDATA.BIN to be loaded into memory offset by $2000 from its normal load address. If XDATA normally resided at $0100, the new location would be $2100. Remember that no code is modified, so unless the binary file is relocatable code, it will not run in its new location.

PRUL

The PRUL command is used to translate upper case only Text Processor files (those containing the 'cap' @ and ^ capitilization characters) into upper and lower case files. This is useful for converting old formatted text files.

DESCRIPTION

The general syntax of the PRUL command is:

PRUL,<input file spec>,<output file spec>

where the file specs default to TXT extensions and to the working drive. The name given to the output file must not already exist. The input file is read and all letters are converted to lower case unless preceeded by an at-sign (@) or if surrounded by up-arrows (^). These are the same rules the Text Processor follows while in the capitalization mode. An example follows:

+++PRUL,BOOKREV,BOOKREVL

This command line would cause the file named BOOKREV.TXT to be read, the appropriate translation performed, and written back out to a file named BOOKREVL.TXT. The original file is left unchanged.

PDEL

The PDEL command is a prompting delete utility. Either all files or only files matching a specified match list are displayed by name, one at a time, giving the option of deleting the file or keeping it. This command is very convenient for quickly removing a lot of no longer needed files from a disk.

DESCRIPTION

The general syntax of the PDEL command is:

```
PDEL[,<drive list>][,<match list>]
```

where drive list and match list are the same as described in the CAT command. Upon execution of PDEL, each file name will be printed at the terminal with the question of deleting it:

```
DELETE "FILE" ?
```

At this time three responses are valid. If a 'N' is typed, the file will be left intact and the next name will be displayed. If a 'Y' is typed, that file will be deleted. This utility DOES NOT ask if you are sure you want the file deleted, so make sure the first time! A carriage return may also be typed in response to the prompt at which time control will return back to FLEX. An example follows:

```
+++PDEL,1,.TXT
```

This command line would cause each file on drive 1 which has a TXT extension to be displayed and the delete option offered. Remember that once 'Y' has been typed to the prompt, that file is gone forever!

REPLACE

The REPLACE command will effectively replace one file on a disk by another, deleting the first file. This command is simply a time saver.

DESCRIPTION

The general syntax of the REPLACE command is:

```
REPLACE,<file spec 1>,<file spec 2>
```

where file spec 1 is the name of the file to be deleted and file spec 2 will be renamed to that of file spec 1. The file specs default to a TXT extension and to the working drive. This command has the same affect as the two command sequence:

```
+++DELETE,FILE1
+++RENAME,FILE2,FILE1
```

which effectively replaces FILE1 with FILE2. This can be performed with REPLACE as follows:

```
+++REPLACE,FILE1,FILE2
```

which will cause FILE1.TXT to be deleted and FILE2.TXT to be renamed FILE1.TXT.

RPT

The RPT command allows a command line to be repeatedly executed a specified number of times. This can be useful in diagnostic or demonstration applications.

DESCRIPTION

The general syntax of the RPT command is:

```
RPT,<repeat count>,<any command line>
```

where repeat count specifies the number of times the following command line should be executed. The command line may contain any FLEX utility except RPT and may also contain multiple commands by using the TTYSET End of Line character. An example follows:

```
+++RPT,6,LIST,BOOK4
```

This line would cause the file BOOK4 to be listed 6 times. Simple repeated demonstrations may be set up using the RPT command.

~ PPT, 10, MEMBLIST = 10 × REPEATING MEMBLIST

REMSPC

The REMSPC command will remove all excess spaces from a text file. This is a useful utility for file conversions and file space reduction.

DESCRIPTION

The general syntax of the REMSPC command is:

REMSPC,<input file spec>,<output file spec>

where the file specs default to a TXT extension and to the working drive. The input file is the file to be processed and the output file name must not already exist on the disk. REMSPC will convert all occurences of two or more spaces into a single space unless the line starts with an asterisk (*) in column 1 (a comment line). These comment lines are passed unmodified to the output file. An example of using REMSPC follows:

+++REMSPC,SOURCE,SOURCE2

This would cause the file named SOURCE.TXT to be read from the working drive, all excess spaces removed, and written back to a file named SOURCE2.TXT on the working drive.

RECOVER

The RECOVER command allows the recovery of a file from a disk whose directory has been damaged. It is necessary to know the starting disk address of the file to be recovered.

DESCRIPTION

The general syntax of the RECOVER command is:

RECOVER,<disk address>,<file spec>

where the file spec defaults to a TXT extension and is forced to drive 0. The bad disk must be in drive 1 and the good disk in drive 0. The disk address is specified as a 4 digit hex number, such that the address track 3, sector 10, would be specified as 030A. This routine starts reading the file on drive 1 at the specified address and copies it into a file on drive 0 giving it the designated name. An example follows:

+++RECOVER,1103,INVEN

This would start reading data from drive 1 at track 17 (hex 11), sector 3, and copy it into a file named INVEN.TXT on drive 0. This process continues until an end of file is encountered.

SPLIT

The SPLIT command is used to split a text file into two new files at a specified line number. It is convenient to use when a file becomes too large to easily manage or to break off an often used section of text into another file.

DESCRIPTION

The general syntax of the SPLIT command is:

SPLIT,<input file spec>,<out file spec1>,<out file spec2>,<N>

The input file is the file to be split, output file spec 1 is the name to be assigned to the first set of lines read from the input file, output spec 2 is the name to be assigned to the rest of the file being split, and N is the line number where the file should be split. The second output file will begin with line N of the input file. All files default to TXT extensions and to the working drive. An example follows:

+++SPLIT,TEST,TEST1,TEST2,125

This command line would cause lines 1 to 124 of the file named TEST.TXT on the working drive to be written into a file named TEST1.TXT and lines 125 to the end of the file to be written into a file named TEST2.TXT. The original file (TEST) remains unchanged.

SLEEP

The SLEEP command is used to eat up time. It is handy to use in special applications in a EXEC command file.

DESCRIPTION

The general syntax of the SLEEP command is:

```
SLEEP,N
```

where N is the number of seconds the system is to "sleep". It should be noted that large values of N (greater than 15) may not timeout exactly. Once the SLEEP command is executed, it cannot be interrupted. You must wait for the entire time period specified. An example will demonstrate SLEEP's use.

```
+++SLEEP,10
```

This command line would cause the system to lockup or sleep for approximately 10 seconds.

TYPOS

The TYPOS utility is used for grouping and counting all words used in a text file. It is a great aid in detecting misspelled words and typographical errors, as well as pointing out too often used words in a document.

DESCRIPTION

The general syntax of the TYPOS command is:

```
TYPOS,<file spec>[,<count>]
```

where <count> specifies the highest word use count to be listed in the final word list. The file spec defaults to a TXT extension and to the working drive. If the count is not specified, the default will be 3, so words appearing three times or less will be listed. All letters are mapped to lower case so words like 'Test' and 'test' would be considered identical. The final word list is printed with each word preceeded by its occurence count and the word in lower case letters. The more often used words are printed first. Following are a few examples.

```
+++TYPOS,CHAPTER1
+++TYPOS,BOOK,50
```

The first example would print all words occuring three times or less found in the file named CHAPTER1.TXT on the working drive. The second example would print all words occuring 50 times or less in the file BOOK.TXT. It should be noted that on long files, TYPOS may require a moderate amount of time to compile the list of words.

TEST

The TEST command is used for testing all sectors on a diskette. Any bad sectors found will be reported to the terminal.

DESCRIPTION

The general syntax of the TEST command is:

```
TEST[,<drive number>]
```

where the drive number specifies which disk is to be tested and defaults to the working drive. Any sectors found to be bad during the test are reported to the terminal in the form of two hex numbers, the first representing the track number, the second is the sector. An example follows:

```
+++TEST,0
```

This will test the diskette in drive 0 for bad sectors. It should be noted that TEST requires a moderat amount of time to run since all data on the disk is read.

MEMFILL

The MEMFILL command is used to fill a section of memory with a particular data pattern. This is useful for certain types of program debugging and development.

DESCRIPTION

The general syntax of the MEMFILL command is:

```
MEMFILL,<start address>,<end address>,<fill byte>
```

The addresses should be specified in hex. The fill byte is the hex value (8 bit) which will be used to fill memory between the address bounds designated. If the fill byte is left off the command line, zeroes will be used. Upon completion, control will return to FLEX unless it has been overwritten by the command. An example follows:

```
+++MEMFILL,0,1FFF,55
```

This would fill memory from location 0 through location hex 1FFF with hex 55 bytes. Remember not to overwrite the program when specifying the address bounds.

UP-LOW

The UP-LOW command is used to convert a file into all lower case letters. It is useful for those systems unable to work with upper case letters or to make a file easier to read.

DESCRIPTION

The general syntax of the UP-LOW command is:

UP-LOW,<input file spec>,<output file spec>

The input file spec specifies the name of the file needing the conversion, and the output file spec specifies the file name of the new converted file. Both default to a TXT extension and to the working drive. The new file will end up containing only lower case letters and the original file will be left unchanged. An example follows:

+++UP-LOW,TEXTA,TEXTAL

This would cause a file named TEXTAL.TXT to be created which is identical to the file named TEXTA.TXT except all letters will be lower case.

WORDS

The WORDS utility is used to get a total word and line count of a text file. It is very useful in document and report preparation in keeping track of the size of the file.

DESCRIPTION

The general syntax of the WORDS command is:

```
WORDS,<file spec>
```

where the file spec defaults to a TXT extension and to the working drive. Upon execution, WORDS will read the file specified, count all words and lines, and then report the totals to the terminal. A word is considered to be any group of characters separated by either spaces or carriage returns. An example will demonstrate the use of WORDS.

```
+++WORDS,CHAPTER1
```

This command line would cause all of the words of the file named CHAPTER1.TXT on the working drive to be counted and the totals diplayed on the terminal.