The FLEX Disk Operating System

GENERAL INFORMATION

THE FLEX DISK OPERATING SYSTEM

I. INTRODUCTION

The FLEX* Operating System is a very versatile and flexible operating system. It provides the user with a powerful set of system commands to control all disk operations directly from the user's terminal. The systems programmer will be delighted with the wide variety of disk access and file management routines available for personal use. Overall, FLEX is one of the most powerful operating systems available today.

The FLEX Operating System is comprised of three parts, the File Management System (FMS), the Disk Operating System (DOS), and the Utility Command Set (UCS). Part of the power of the overall system lies in the fact that the system can be greatly expanded by simply adding additional utility commands. The user should expect to see many more utilities available for FLEX in the future. Some of the other important features include: fully dynamic file space allocation, the automatic "removal" of defective sectors from the disk, automatic space compression and expansion on all text files, complete user environment control using the TTYSET utility command, and uniform disk wear due to the high performance dynamic space allocator.

The UCS currently contains many very useful commands. These programs reside on the system disk and are only loaded into memory when needed. This means that the set of commands can be easily extended at any time, without the necessity of replacing the entire operating system. The utilities provided with FLEX perform such tasks as the saving, loading, copying, renaming, deleting, appending, and listing of disk files. There is an extensive CATalog command for examining the disk's file directory. Several environment control commands are also provided. Overall, FLEX provides all of the necessary tools for the user's interaction with the disk.

----

* FLEX is a registered trademark of Technical Systems Consultants, Inc.

II. SYSTEM REQUIREMENTS

FLEX requires random access memory from location 0000 through location 2FFF hex (12K). Memory is also required from C000 (48K) through DFFF hex (56K), where the actual operating system resides. The system also assumes at least 2 disk drives are connected to the controller and that they are configured as drives #0 and #1. You should consult the disk drive instructions for this information. FLEX interfaces with the disk controller through a section of driver routines and with the operator console or terminal through a section of terminal I/O routines.

III. GETTING THE SYSTEM STARTED

Each FLEX system diskette contains a binary loader for loading the operating system into RAM. There needs to be some way of getting the loader off of the disk so it can do its work. This can be done by either hand entering the bootstrap loader provided with the disk system, or by using the boot provided in ROM if appropriate to FLEX.

As a specific example, suppose the system we are using has SWTPc's S-BUG installed and we wish to run FLEX. The first step is to power on all equipment and make sure the S-BUG prompt is present (>). Next insert the system diskette into drive 0 (the boot must be performed with the disk in drive 0) and close the door on the drive. Type "D" on the terminal if using a full size floppy system or "U" if a minifloppy system. The disk motors should start, and after about 2 seconds, the following should be displayed on the terminal:

```
FLEX X.X
DATE (MM,DD,YY)?

+++
```

The name FLEX identifies the operating system and the X.X will be the version number of the operating system. At this time the current date should be entered, such as 7,3,79. The FLEX prompt is the three plus signs (+++), and will always be present when the system is ready to accept an operator command. The '+++' should become a familiar sight and signifies that FLEX is ready to work for you!

IV. DISK FILES AND THEIR NAMES

All disk files are stored in the form of 'sectors' on the disk and in this version, each sector contains 256 'bytes' of information. Each byte can contain one character of text or one byte of binary machine information. A maximum of 340 user-accessible sectors will fit on a single-sided mini disk or 1140 sectors on a single-sided full size floppy. Double-sided disks would hold exactly twice that number of sectors. Double-density systems will hold more still. The user, however, need not keep count, for the system does this automatically. A file will always be at least one sector long and can have as many as the maximum number of sectors on the disk. The user should not be concerned with the actual placement of the files on the disk since this is done by the operating system. File deletion is also supported and all previously used sectors become immediately available again after a file has been deleted.

All files on the disk have a name. Names such as the following are typical:

```
PAYROLL
INVENTORY
TEST1234
APRIL-78
WKLY-PAY
```

Anytime a file is created, referenced, or deleted, its name must be used. Names can be most anything but must begin with a letter (not numbers or symbols) and be followed by at most 7 additional characters, called 'name characters'. These 'name characters' can be any combination of the letters 'A' through 'Z' or 'a' through 'z', any digit '0' through '9', or one of the two special characters, the hyphen (-) or the underscore '_', (a left arrow on some terminals).

File names must also contain an 'extension'. The file extension further defines the file and usually indicates the type of information contained therein. Examples of extensions are: TXT for text type files, BIN for machine readable binary encoded files, CMD for utility command files, and BAS for BASIC source programs. Extensions may contain up to 3 'name characters' with the first character being a letter. Most of the FLEX commands assume a default extension on the file name and the user need not be concerned with the actual extension on the file. The user may at anytime assign new extensions, overiding the default value, and treat the extension as just part of the file name. Some examples of file names with their extensions follow:

```
APPEND.CMD
LEDGER.BAS
TEST.BIN
```

Note that the extension is always separated from the name by a period '.'. The period is the name 'field separator'. It tells FLEX to treat the following characters as a new field in the name specification.

A file name can be further refined. The name and extension uniquely define a file on a particular drive, but the same name may exist on several drives simultaneously. To designate a particular drive a 'drive number' is added to the file specification. It consists of a single digit (0-3) and is separated from the name by the field separator '.'. The drive number may appear either before the name or after it (after the extension if it is given). If the drive is not specified, the system will default to either the 'system' drive or the 'working' drive. These terms will be described a little later.

Some examples of file specifications with drive numbers follow:

```
0.BASIC
MONDAY.2
1.TEST.BIN
LIST.CMD.1
```

In summary, a file specification may contain up to three fields separated by the field separator. These fields are; 'drive', 'name', and 'extension'. The rules for the file specification can be stated quite concisely using the following notation:

```
[<drive>.]<name>[.<extension>]
  or
<name>[.<extension>][.<drive>]
```

The '<>' enclose a field and do not actually appear in the specification, and the '[]' surround optional items of the specification. The following are all syntactically correct:

```
0.NAME.EXT
NAME.EXT.0
NAME.EXT
0.NAME
NAME.0
NAME
```

Note that the only required field is the actual 'name' itself and the other values will usually default to predetermined values. Studying the above examples will clarify the notation used. The same notation will occur regularly throughout the manual.

V. ENTERING COMMANDS

When FLEX is displaying '+++', the system is ready to accept a command line. A command line is usually a name followed by certain parameters depending on the command being executed. There is no 'RUN' command in FLEX. The first file name on a command line is always loaded into memory and execution is attempted. If no extension is given with the file name, 'CMD' is the default. If an extension is specified, the one entered is the one used. Some examples of commands and how they would look on the terminal follow:

```
+++TTYSET
+++TTYSET.CMD
+++LOOKUP.BIN
```

The first two lines are identical to FLEX since the first would default to an extension of CMD. The third line would load the binary file 'LOOKUP.BIN' into memory and, assuming the file contained a transfer address, the program would be executed. A transfer address tells the program loader where to start the program executing after it has been loaded. If you try to load and execute a program in the above manner and no transfer address is present, the message, 'NO LINK' will be output to the terminal, where 'link' refers to the transfer address. Some other error messages which can occur are 'WHAT?' if an illegal file specification has been typed as the first part of a command line, and 'NOT THERE' if the file typed does not exist on the disk.

During the typing of a command line, the system simply accepts all characters until a 'RETURN' key is typed. Any time before typing the RETURN key, the user may use one of two special characters to correct any mistyped characters. One of these characters is the 'back space' and allows deletion of the previously typed character. Typing two back spaces will delete the previous two characters. The back space is initially defined to be a 'control H' but may be redefined by the user using the TTYSET utility command. The second special character is the line 'delete' character. Typing this character will effectively delete all of the characters which have been typed on the current line. A new prompt will be output to the terminal, but instead of the usual '+++' prompt, to show the action of the delete character, the prompt will be '???'. Any time the delete character is used, the new prompt will be '???', and signifies that the last line typed did not get entered into the computer. The delete character is initially a 'control X' but may also be redefined using TTYSET.

As mentioned earlier, the first name on a command line is always interpreted as a command. Following the command is an optional list of names and parameters, depending on the particular command being entered. The fields of a command line must be separated by either a space or a comma. The general format of a command line is:

```
     <command>[,<list of names and parameters>]
```

A comma is shown, but a space may be used. FLEX also allows several commands to be entered on one command line by use of the 'end of line' character. This character is initially a colon (':'), but may be user defined with the TTYSET utility. By ending a command with the end of line character, it is possible to follow it immediately with another command. FLEX will execute all commands on the line before returning with the '+++' prompt. An error in any of the command entries will cause the system to terminate operation of that command line and return with the prompt. Some examples of valid command lines follow:

```
     +++CAT 1
     +++CAT 1:ASN S=1
     +++LIST LIBRARY:CAT 1:CAT 0
```

As many commands may be typed in one command line as desired, but the total number of characters typed must not exceed 128. Any excess characters will be ignored by FLEX.

One last system feature to be described is the idea of 'system' and 'working' drives. As stated earlier, if a file specification does not specifically designate a drive number, it will assume a default value. This default value will either be the current 'system' drive assignment or the current 'working' drive assignment. The system drive is the default for all command names, or in other words, all file names which are typed first on a command line. Any other file name on the command line will default to the working drive. This version of FLEX also supports automatic drive searching. When in the auto search mode if no drive numbers are specified, the operating system will first search drive 0 for the file. If the file is not found, drive 1 will be searched and so on. When the system is first initialized the auto drive searching mode will be selected. At this time, all drive defaults will be to drive 0. It is sometimes convenient to assign drive 1 as the working drive in which case all file references, except commands, will automatically look on drive 1. It is then convenient to have a diskette in drive 0 with all the system utility commands on it (the 'system drive'), and a disk with the files being worked on in drive 1 (the 'working drive'). If the system drive is 0 and the working drive is 1, and the command line was:

```
     +++LIST TEXTFILE
```

FLEX would go to drive 0 for the command LIST and to drive 1 for the file TEXTFILE. The actual assignment of drives is performed by the ASN utility. See its description for details.

VI. COMMAND DESCRIPTIONS

There are two types of commands in FLEX, memory resident (those which actually are part of the operating system) and disk utility commands (those commands which reside on the disk and are part of the UCS).

There are four memory resident commands included in FHL Color FLEX: GET, MON, ROM, and P. Even though they are not considered part of the Disk Utility Command Set, for simplicities sake they will be discussed in the UCS section of this manual.

GENERAL SYSTEM INFORMATION

I. DISK CAPACITY

Each sector of a FLEX disk contains 252 characters or bytes of user date (4 bytes of 256 byte sector are used by the system). Thus a single-sided mini disk has 340 sectors or 85,680 characters or bytes of user information. A single-sided full size disk has 1140 sectors or 287,280 bytes of user data. Double-sided disks would contain exactly twice these amounts.

II. WRITE PROTECT

Floppy disks can usually be physically write protected to prevent FLEX from performing a write operation. Any attempt to write to such a disk will cause an error message to be issued. It is good practice to write protect disks which have important files on them.

A mini disk can be write protected by placing a piece of opaque tape over the small rectangular cutout on the edge of the disk. Full size floppys are just the opposite. In order to write protect a full size disk, you must remove the tape from the cutout. In other words, the notch must be exposed to write protect the disk. Some full size disks do not have this cutout, and therefore cannot be write protected.

III. THE 'RESET' BUTTON

The RESET button on the back panel of your computer should NEVER BE PRESSED DURING A DISK OPERATION. There should never be a need to reset the machine while in FLEX. If the machine is reset and the system is writing data on the disk, it is possible that the entire disk will become damaged. Again, never press 'reset' while the drive is operating! Refer to the 'escape' character in TTYSET for ways of stopping FLEX.

IV. NOTES ON THE 'P' COMMAND

On a standard FLEX system, the 'P' command tries to load a printer driver file named 'PRINT.SYS' from the same disk which P itself was retrieved (after first making sure that a driver has not already been loaded in). FHL Color FLEX utilizes P in a different manner:

P is a memory resident command which simply redirects I/O to the printer port of the Color Computer. Instead of playing the part of the hotel doorman (the function of the standard P) calling a taxi (PRINT.SYS) for the waiting patron (the file to be output), P in this case takes the patron directly to a location where he knows the "hotel taxi" is waiting. This particular hotel is not serviced by any other taxis, so the doorman (P) knows there is only one way out - the hotel taxi.

As presently is and has been the case on the Color Computer, there is only one output port - the RS 232 port. Similar to the hotel with the "built-in" taxi, FHL Color FLEX has a built in (memory-resident) printer driver just sitting there at the RS232 port waiting for P to send it some data to be printed. The reason it can be built in (memory-resident) is because there is only need for one standard driver since all data on the Color Computer is being sent out one standard port (the RS 232).

For more information on the standard FLEX PRINT.SYS file, see page 53 of the 'Advanced Programmer's Guide.'

VI. SYSTEM ERROR NUMBERS

Any time that FLEX detects an error during an operation, an appropriate error message will be displayed on the terminal. FLEX internally translates a derived error number into a plain language statement using a look-up table called ERROR.SYS. If you have forgotten to copy this .SYS file onto a disk that you are using, FLEX will report a corresponding number as shown below:

```
DISK ERROR  #xx
```

where 'xx' is a decimal error number. The table below is a list of these numbers and what error they represent.

ERROR #	MEANING
1	ILLEGAL FMA FUNCTION CODE ENCOUNTERED
2	THE REQUESTED FILE IS IN USE
3	THE FILE SPECIFIED ALREADY EXISTS
4	THE SPECIFIED FILE COULD NOT BE FOUND
5	SYSTEM DIRECTORY ERROR-REBOOT SYSTEM
6	THE SYSTEM DIRECTORY IS FULL
7	ALL AVAILABLE DISK SPACE HAS BEEN USED
8	READ PAST END OF FILE
9	DISK FILE READ ERROR
10	DISK FILE WRITE ERROR
11	THE FILE OR DISK IS WRITE PROTECTED
12	THE FILE IS PROTECTED-FILE NOT DELETED
13	ILLEGAL FILE CONTROL BLOCK SPECIFIED
14	ILLEGAL DISK ADDRESS ENCOUNTERED
15	AN ILLEGAL DRIVE NUMBER WAS SPECIFIED
16	DRIVE NOT READY
17	THE FILE IS PROTECTED-ACCESS DENIED
18	SYSTEM FILE STATUS ERROR
19	FMS DATA INDEX RANGE ERROR
20	FMS INACTIVE-REBOOT SYSTEM
21	ILLEGAL FILE SPECIFICATION
22	SYSTEM FILE CLOSE ERROR
23	SECTOR MAP OVERFLOW-DISK TOO SEGMENTED
24	NON-EXISTENT RECORD NUMBER SPECIFIED
25	RECORD NUMBER MATCH ERROR-FILE DAMAGED
26	COMMAND SYNTAX ERROR-RE-TYPE COMMAND
27	THAT COMMAND IS NOT ALLOWED WHILE PRINTING
28	WRONG HARDWARE CONFIGURATION

For more details concerning the meanings of these error messages, consult the 'Advanced Programmer's Guide'(pgs. 38 - 40).

VII. FLEX OPERATING SYSTEM INPUT/OUTPUT SUBROUTINES

In order for the FLEX I/O functions to operate properly, all user program character input/output subroutines should be vectored thru the FLEX operating system rather than the computer's monitor. Below is a list of FLEX's I/O subroutines and a brief description of each. All given addresses are in hexadecimal.

GETCHR at $CD15

This subroutine is functionally equivalent to S-BUG's character input routine. This routine will look for one character from the control terminal (I/O port #1) and store it in the A accumulator. Once called, the input routine will loop within itself until a character has been input. Anytime input is desired, the call JSR GETCHR or JSR $CD15 should be used.

GETCHR automatically sets the 8th bit to 0 and does not check for parity. A call to this subroutine affects the processor's registers as follows:

ACC. A	loaded with the character input from the terminal
B,X,Y,U	not affected

PUTCHR at $CD18

This subroutine is used to output one character from the computer to the control port (I/O port #1). It is functionally equivalant to the output character routine in S-BUG.

To use PUTCHR, the character to be output should be placed in the A accumulator in its ASCII form. For example, to output the letter 'A' on the control terminal, the following program should be used:

```
LDA    #$41
JSR    $CD18
```

The processor's registers are affected as follows:

ACC. A	changed internally
B,X,Y,U	not affected

PSTRNG at $CD1E

PSTRNG is a subroutine used to output a string of text on the control terminal. When address $CD1E is called, a carriage return and line feed will automatically be generated and data output will begin at the location pointed to by the index register. Output will continue until a hex 04 is seen. The same rules for using the ESCAPE and RETURN keys for stopping output apply as described earlier.

The accumulator and register status after using PSTRNG are as follows:

ACC. A	Changed during the operation

ACC. B	Unchanged
X	Contains the memory location of the last character read from the string (usually the 04 unless stopped by the ESC key)
Y,U	Unchanged

NOTE: The ability of using backspace and line delete characters is a function of your user program and not of the FLEX I/O routines described above.

For additional information consult the 'Advanced Programmer's Manual'.

STAT at $CD4E
This routine is used to determine the "status" of the input device. That is, to see if a character has been typed on the input terminal keyboard. Its function is to check for characters such as the ESCAPE key in FLEX which allows breaking of the output. This routine returns an EQual condition if no character was hit and a Not-Equal condition if a character was hit. No registers, except for the condition codes, may be altered.

VIII. BOOTING THE FLEX DISK OPERATING SYSTEM

In order to read FLEX from the system disk upon powering up your system, you must have a short program in RAM or ROM memory. This program is called a 'bootstrap' loader.

With FHL Color FLEX, typing in RUN"FLEX" will call the BASIC bootstrap loader called "FLEX".

Non-Color Computer Users should use the boot supplied with the hardware if compatible with FLEX. A sample boot (for the SWTPc mini system) is given here for reference.

If the system does not boot properly, re-position the system disk in the drive and re-execute the bootstrap loader.

```
0100 B6   E018    START   LDA    COMREG    TURN MOTOR ON
0103 86   00              LDA    #0
0105 B7   E014            STA    DRVREG
0108 8E   0000            LDX    #0000
010B 3D           OVR     MUL              DELAY FOR SPEED UP
010C 30   1F              LEAX   -1,X
010E 26   FB              BNE    OVR
0110 C6   0F              LDB    #$0F      RESTORE
0112 F7   E018            STB    COMREG
0115 8D   2B              BSR    RETURN
0117 F6   E018    LOOP1   LDB    COMREG
011A C5   01              BITB   #1
011C 26   F9              BNE    LOOP1
011E 86   01              LDA    #1
0120 B7   E01A            STA    SECREG
0123 8D   1D              BSR    RETURN
0125 C6   8C              LDB    #$8C      READ WITH LOAD
0127 F7   E018            STB    COMREG
012A 8D   16              BSR    RETURN
012C 8E   C000            LDX    #$C000
012F C5   02      LOOP2   BITB   #2        DRQ?
0131 27   05              BEQ    LOOP3
0133 B6   E01B            LDA    DATREG
0136 A7   80              STA    0,X+
0138 F6   E018    LOOP3   LDB    COMREG
013B C5   01              BITB   #1        BUSY?
013D 26   F0              BNE    LOOP2
013F 7E   C000            JMP    $C000
0142 8D   00      RETURN  BSR    RTN
0144 39           RTN     RTS
```

IX. REQUIREMENTS FOR THE 'PRINT.SYS' PRINTER DRIVER

(This does not apply to FHL Color FLEX. See pg. 1.9)

Standard TSC FLEX, as supplied, includes a printer driver that will work with most parallel type printers, such as the SWTPC PR-40. If desired, the printer driver may be changed to accomodate other types of printers. Included is the source listing for the supplied driver. Additional information on the requirements for the PRINT.SYS driver can be found in the 'Advanced Programmer's Guide' on page 53.

1) The driver must be in a file called PRINT.SYS

2) Three separate routines must be supplied, a printer initialization routine (PINIT at $CCC0), a check ready routine (PCHK at $CCD8), and an output character routine (POUT at $CCE4).

3) When the POUT routine is called by FLEX, the character to be output will be in the A accumulator. The output routine must not destroy the B, X, Y, or U registers. PINIT may destroy any registers. PCHK may NOT alter any registers.

4) The routines MUST start at the addresses specified, but may be continued anywhere in memory if there is not room where specified. If placed elsewhere in memory, be certain they do not conflict with any utilities or programs which will use them.

5) All three routines must end with a return from subroutine instruction (RTS).

```
                         *
                         * PRINT.SYS PIA DRIVERS FOR GENERAL CASE PRINTER
                         *

                   E01C  PIA     EQU    $E01C     PIA ADDRESS FOR PORT #7

                         *
                         * PRINTER INITIALIZATION (MUST BE AT $CCC0)
                         *
CCC0                             ORG    $CCC0     MUST RESIDE AT $CCC0
CCC0 86    3A            PINIT   LDA    #$3A      SELECT DATA DIRECTION REG.
CCC2 B7    E01D                  STA    PIA+1     BY WRITING 0 IN DDR CONTROL
CCC5 86    FF                    LDA    #$FF      SELECT ALL OUTPUT LINES
CCC7 B7    E01C                  STA    PIA       PUT IN DATA DIRECTION REG.
CCCA 86    3E                    LDA    #$3E      SET UP FOR TRANSITION CHECKS
CCCC B7    E01D                  STA    PIA+1     AND ENABLE OUTPUT REGISTER
CCCF 39                          RTS
                         * PRINTER READY ROUTINE
CCD0 7D    E01C          PREADY  TST    PIA       RESET PIA READY INDICATION
CCD3 73    CCE3                  COM    PFLAG     SET THE PRINTER READY FLAG
CCD6 39                          RTS
```

```
                         *
                         * CHECK FOR PRINTER READY (MUST BE AT $CCD8)
                         *
CCD8                             ORG    $CCD8      PRINT TEST AT $CCD8
CCD8 7D    CCE3          PCHK    TST    PFLAG      TEST FOR PRINTER READY
CCDB 2B    05                    BMI    PCHKX      IF NEGATIVE, PRINTER READY
CCDD 7D    E01D                  TST    PIA+1      CHECK FOR TRANSITION
CCE0 2B    EE                    BMI    PREADY     IF MINUS, PRINTER NOW READY
CCE2 39                  PCHKX   RTS
                         * PRINTER READY FLAG
CCE3 FF                  PFLAG   FCB    $FF        PRINTER READY FLAG

                         *
                         * PRINTER OUTPUT CHARACTER ROUTINE (MUST BE AT $CCE4)
                         *
CCE4                             ORG    $CCE4      MUST RESIDE AT $CCE4
CCE4 8D    F2            POUT    BSR    PCHK       TEST FOR PRINTER READY
CCE6 2A    FC                    BPL    POUT       LOOP UNTIL PRINTER READY
CCE8 7F    CCE3                  CLR    PFLAG      SET PRINTER FLAG NOT READY
CCEB B7    E01C                  STA    PIA        SET DATA IN OUTPUT REGISTER
CCEE 86    36                    LDA    #$36       SET DATA READY, HIGH TO LOW
CCF0 8D    02                    BSR    POUTB      STUFF BYTE INTO THE PIA
CCF2 86    3E                    LDA    #$3E       THEN SEARCH FOR TRANSITION
CCF4 B7    E01D          POUTB   STA    PIA+1      OF LOW LEVEL TO HIGH LEVEL
CCF7 39                          RTS

                                 END
```