FLEX
For the TRS-80 Color Computer

The
UTILITY
COMMAND
SET

Technical Systems Consultants, Inc.
P.O. Box 2570
West Lafayette, Indiana 47906

FLEX
For the TRS-80 Color Computer

The
UTILITY
COMMAND
SET

Technical Systems Consultants, Inc.
P.O. Box 2570
West Lafayette, Indiana 47906

(*) These commands are provided by Frank Hogg Laboratory, Inc., not TSC. Contact FHL if you have any problems with them.

NOTE: IF THERE IS A 'READ-ME.TXT' FILE ON THE SUPPLIED DISK THEN LIST IT OUT USING:

```
LIST 0.READ-ME
```

(*) These commands are provided by Frank Hogg Laboratory, Inc. - not TSC. Contact FHL if you have any problems with them.

PLEASE NOTE: The following documentation pertains to Standard TSC FLEX. In adapting FLEX to run on the Color Computer, the following changes were made. Please be aware of these differences when reading the following documentation.

1. The 'P' command in FHL Color FLEX has a different meaning than that referenced in the following section. See 'Notes on the P Command' page 1.9.

2. Printer Spooling is not supported.

3. Interrupts are not used.

PLEASE NOTE: The following documentation pertains to Standard TSC FLEX. In adapting FLEX to run on the Color Computer, the following changes were made. Please be aware of these differences when reading the following documentation.

1. The 'P' command in FHL Color FLEX has a different meaning than that referenced in the following section. See 'Notes on the P Command' page 1.9.

2. Printer Spooling is not supported.

3. Interrupts are not used.

APPEND,<file spec>[,<file list>],<file spec>
   Default extension: .TXT
   Description page: A.1.1

ASN[,W=<drive>][,S=<drive>]
   Default: W=0,S=0
   Description page: A.2.1

BASIC
   Description page: C.3.1

BUILD,<file spec>
   Default extension: .TXT
   Description page: B.1.1

CAT[,<drive list>][,<match list>]
   Description page: C.1.1

CBASIC
   Description page: C.2.1

COPY,<file spec>,<file spec>
COPY,<file spec>,<drive>
COPY,<drive>,<drive>[,<match list>]
   Description page: C.3.1

DATE[,<MM,DD,YY>]
   Description page: D.1.1

DBASIC
   Description page: D.2.1

DELETE,<file spec>[,<file list>]
   Description page: D.3.1

DISPLAY,<list>
   Description page: D.4.1

EXEC,<file spec>
   Default extension: .TXT
   Description page: E.1.1

EXT
   Description page: E.2.1

GET,<file spec>[,<file list>]
   Description page: 1.7

HELP
   Description page: H.1

I,<file spec>,<command>
   Default extension: .TXT
   Description page: I.1.1

INT
   Description page: I.2.1

ISM
   Description page: I.3.1

JUMP,<hex address>
   Description page: J.1.1

LINK,<file spec>
   Default extension: .SYS
   Description page: L.1.1

LIST,<file spec>[,<line range>][,N]
   Default extension: .TXT
   Description page: L.2.1

MON
   Description page: M.1.1

MOVEROM
   Description page: M.2.1

N,<command>
   Description page: N.2.1

NEWDISK,<drive>
   Description page: N.1.1

NEWDISKA,<drive>
   Description page: N.1.1

O,<file spec>,<command>
   Default extension: .OUT
   Description page: O.1.1

P,<command>
   Description page: P.1.1

PROT,<file spec>[,<options>]
   Description page: P.2.1

PUTBOOT.LDR,<drive number>
   Description page: P.3.1

RENAME,<file spec 1>,<file spec 2>
   Default extension: .TXT
   Description page: R.1.1

COMMAND SUMMARY

ROM
Description page: R.2.1

SAVE,<file spec>,<begin adr>,<end adr>[,<transfer adr>]
Default extension: .BIN
Description page: S.1.1

SDC
Description page: S.2.1

SETUP,<option field>[,<option field>...]
Description page: S.3.1

STARTUP
Description page: S.4.1

TED
Description page: T.2.1

TTYSET[,<parameter list>]
Description page: T.1.1

VERIFY[,<file spec>]
Default extension: .CMD
Description page: V.1.1

VERSION,<file spec>
Default extension: .CMD
Description page: V.2.1

XOUT[,<drive spec>]
Description page: X.1.1

Xscreens
Description page: X.2.1

Y,<command>
Description page: Y.1.1

UTILITY COMMAND SET

The following pages describe all of the utility commands currently included in the UCS. You should note that the page numbers denote the first letter of the command name, as well as the number of the page for a particular command. For example, 'B.1.2' is the 2nd page of the description for the 1st utility name starting with the letter 'B'.

COMMON ERROR MESSAGES

Several error messages are common to many of the FLEX utility commands. These error messages and their meanings include the following:

NO SUCH FILE. This message indicates that a file referenced in a particular command was not found on the disk specified. Usually the wrong drive was specified (or defaulted), or a misspelling of the name was made.

ILLEGAL FILE NAME. This can happen if the name or extension did not start with a letter, or the name or extension field was too long (limited to 8 and 3 respectively). This message may also mean that the command being executed expected a file name to follow and one was not provided.

FILE EXISTS. This message will be output if you try to create a file with a name the same as one which currently exists on the same disk. Two different files with the same name are not allowed to exist on the same disk.

SYNTAX ERROR. This means that the command line just typed does not follow the rules stated for the particular command used. Refer to the individual command descriptions for syntax rules.

GENERAL SYSTEM FEATURES

Any time one of the utility commands is sending output to the terminal, it may be temporarily halted by typing the 'escape' character (see TTYSET for the definition of this character). Once the output is stopped, the user has two choices: typing the 'escape' character again or typing 'RETURN'. If the 'escape' character is typed again, the output will resume. If the 'RETURN' is typed, control will return to FLEX and the command will be terminated. All other characters are ignored while output is stopped.

APPEND

The APPEND command is used to append or concatenate two or more files, creating a new file as the result. Any type of file may be appended but it only makes sense to append files of the same type in most cases. If appending binary files which have transfer addresses associated with them, the transfer address of the last file of the list will be the effective transfer address of the resultant file. All of the original files will be left intact.

DESCRIPTION

The general syntax for the APPEND command is as follows:

APPEND,<file spec>[,<file list>],<file spec>

where <file list> can be an optional list of the specifications. The last name specified should not exist on the disk since this will be the name of the resultant file. If the last file name given does exist on the disk, the question "MAY THE EXISTING FILE BE DELETED?" will be displayed. A Y response will delete the current file and cause the APPEND operation to be completed. A N response will terminate the APPEND operation. All other files specified must exist since they are the ones to be appended together. If only 2 file names are given, the first file will be copied to the second file. The extension default is TXT unless a different extension is used on the FIRST FILE SPECIFIED, in which case that extension becomes the default for the rest of the command line. Some examples will show its use:

```
APPEND,CHAPTER1,CHAPTER2,CHAPTER3,BOOK
APPEND,FILE1,1.FILE2.BAK,GOODFILE
```

The first line would create a file on the working drive called 'BOOK.TXT' which would contain the files 'CHAPTER1.TXT', CHAPTER2.TXT', and 'CHAPTER3.TXT' in that order. The second example would append 'FILE2.BAK' from drive 1 to FILE1.TXT from the working drive and put the result in a file called 'GOODFILE.TXT' on the working drive. The file GOODFILE defaults to the extension of TXT since it is the default extension. Again, after the use of the APPEND command, all of the original files will be intact, exactly as they were before the APPEND operation.

The ASN command is used for assigning the 'system' drive and the 'working' drive or to select automatic drive searching. The system drive is used by FLEX as the default for command names or, in general, the first name on a command line. The working drive is used by FLEX as the default on all other file specifications within a command line. Upon initialization, FLEX assigns drive #0 as both the system and working drive. An example will show how the system defaults to these values:

```
APPEND,FILE1,FILE2,FILE3
```

If the system drive is assigned to be #0 and the working drive is assigned to drive #1, the above example will perform the following operation: get the APPEND command from drive #0 (the system drive), then append FILE2 from drive #1 (the working drive) to FILE1 from drive #1 and put the result in FILE3 on drive #1. As can be seen, the system drive was the default for APPEND where the working drive was the default for all other file specs listed.

Automatic drive searching causes FLEX to automatically scan the ready drives for the file specified. Hardware limitations prevent the mini floppy versions from searching for "ready" drives. For this reason, FLEX has been setup to ALWAYS assume drive 0 and 1 are ready. Thus if a mini floppy version of FLEX attempts to search a drive which does not have a disk loaded, it will hang up until a disk is inserted and the door closed. Alternatively, the system reset could be hit and a warm start executed (a jump to address $CD03). The full size floppy version CAN detect a ready condition and will not check drives which are out of the ready state during automatic drive searching.

Automatic drive searching causes FLEX to first check drive #0 for the file specified. If not there (or if not ready in the full size version), FLEX skips to drive #1. If the file is not found on drive #1 in the mini floppy version, FLEX gives up and a file not found error results. In the full size version FLEX continues to search on drives #2 and #3 before reporting an error.

DESCRIPTION

The general syntax for the ASN command is as follows:

```
ASN[,W=<drive>][,S=<drive>]
```

where <drive> is a single digit drive number or the letter A. If just ASN is typed followed by a 'RETURN', no values will be changed, but the system will output a message which tells the current assignments of the system and working drives, for example:

```
+++ASN
THE SYSTEM DRIVE IS #0
THE WORKING DRIVE IS #0
```


Some examples of using the ASN command are:

```
ASN,W=1
ASN,S=1,W=0
```

where the first line would set the working drive to 1 and leave the system drive assigned to its previous value. The second example sets the system drive to 1 and the working drive to 0. Careful use of drive assignments can allow the operator to avoid the use of drive numbers on file specifications most of the time!

If auto drive searching is desired, then the letter A for automatic, should be used in place of the drive number.

```
Example:
ASN W=A
ASN S=A, W=1
ASN S=A, W=A
```

BINHEX

+++BINHEX,<binary file>

Used to list a binary file in Motorola S1-S9 format.
After +++O,<output text file>,BINHEX,<binary file> you can edit the S1-S9 records as text.
Using +++P,BINHEX,<binary file> you can download programs to other computers.

BUILD

The BUILD command is provided for those desiring to create small text files quickly (such as STARTUP files, see STARTUP) or not wishing to use the optionally available FLEX Text Editing System. The main purpose for BUILD is to generate short text files for use by either the EXEC command or the STARTUP facility provided in FLEX.

DESCRIPTION

The general syntax of the BUILD command is:

BUILD,<file spec>

where <file spec> is the name of the file you wish to be created. The default extension for the spec is TXT and the drive defaults to the working drive. If the output file already exists the question "MAY THE EXISTING FILE BE DELETED?" will be displayed. A Y response will delete the existing file and build a new file while a N response will terminate the BUILD command.

After you are in the 'BUILD' mode, the terminal will respond with an equals sign ('=') as the prompt character. This is similar to the Text Editing System's prompt for text input. To enter your text, simply type on the terminal the desired characters, keeping in mind that once the 'RETURN' is typed, the line is in the file and can not be changed. Any time before the 'RETURN' is typed, the backspace character may be used as well as the line delete character. If the delete character is used, the prompt will be '???' instead of the equals sign to show that the last line was deleted and not entered into the file. It should be noted that only printable characters (not control characters) may be entered into text files using the BUILD command.

To exit the BUILD mode, it is necessary to type a pound sign ('#') immediately following the prompt, then type 'RETURN'. The file will be finished and control returned back to FLEX where the three plus signs should again be output to the terminal. This exiting is similar to that of the Text Editing System.

* For longer files and additional editing capabilities, see the section on TED (Tiny EDitor).

CAT

The CATalog command is used to display the FLEX disk file names in the directory on each disk. The user may display selected files on one or multiple drives if desired.

DESCRIPTION

The general syntax of the CAT command is:

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

where <drive list> can be one or more drive numbers seperated by commas, and <match list> is a set of name and extension characters to be matched against names in the directory. For example, if only file names which started with the characters 'VE' were to be cataloged, then VE would be in the match list. If only files whose extensions were 'TXT' were to be cataloged, then .TXT should appear in the match list. A few specific examples will help clarify the syntax:

```
+++CAT
+++CAT,1,A.T,DR
+++CAT,PR
+++CAT,0,1
+++CAT,0,1,.CMD,.SYS
```

The first example will catalog all file names on the working drive or on all drives if auto drive searching is selected. The second example will catalog only those files on drive 1 whose names begin with 'A' and whose extensions begin with 'T', and also all files on drive 1 whose names start with 'DR'. The next example will catalog all files on the working drive (or on all drive if auto drive searching is selected) whose names start with 'PR'. The next line causes all files on both drive 0 and drive 1 to be cataloged. Finally, the last example will catalog the files on drive 0 and 1 whose extensions are CMD or SYS.

During the catalog operation, before each drive's files are displayed, a header message stating the drive number is output to the terminal. The name of the diskette as entered during the NEWDISK operation will also be displayed. The actual directory entries are listed in the following form:

```
NAME.EXTENSION    SIZE PROTECTION CODE
```

where size is the number of sectors that file occupies on the disk. If more than one set of matching characters was specified on the command line, each set of names will be grouped according to the characters they match. For example, if all .TXT and .CMD files were cataloged, the TXT types would be listed together, followed by the CMD types.

In summary, if the CAT command is not parameterized, then all files on the assigned working drive will be displayed. If a working drive is not assigned (auto drive searching mode) the CAT command will display files

on all on line drives. If it is parameterized by only a drive number, then all files on that drive will be displayed. If the CAT command is parameterized by only an extension, then only files with that extension will be displayed. If only the name is used, then only files which start with that name will be displayed. If the CAT command is parameterized by only name and extension, then only files of that root name and root extension (on the working drive) will be displayed. Learn to use the CAT command and all of its features and your work with the disk will become a little easier.

The current protection code options that can be displayed are as follows:

D	File is delete protected (delete or rename prohibited)
W	File is write protected (delete, rename and write prohibited)
(blank)	No special protection

CBASIC

The CBASIC command is used to run Radio Shack Extended COLOR BASIC (but not Disk Extended COLOR BASIC) without erasing FLEX from memory.

DESCRIPTION

The general syntax of the CBASIC command is:

CBASIC

This command copies the Radio Shack COLOR BASIC and Exended COLOR BASIC from ROM to RAM at the same address (see MOVEROM). CBASIC then makes some changes to the initialization code of COLOR BASIC so that it will run in RAM. These changes are detailed below.

The user may return to FLEX by typing

EXEC

provided the EXEC pointer of COLOR BASIC has not been changed, for example by using CLOADM. If the EXEC pointer was changed, the user must type

EXEC &HC100

which is the address of the routine used to restart FLEX.

This program does not set up COLOR BASIC to use FLEX I/O. CBASIC uses its own keyboard input and screen output routines. The routine at $C100 reinitializes the FLEX display screen.

The areas of COLOR BASIC changed are as follows:

ADDRESS	DATA	FUNCTION
$A055	$0D	# of SAM locations to reconfigure
$A066	$20	Disable reconfiguring of memory size
$A084	$8E	Disable memory test
$A085	$7F	
$A086	$FE	
$A087	$7E	
$A088	$A0	
$A089	$93	
$A11B	$C1	Set up EXEC pointer
$A11C	$00	

BASIC

BASIC is the same as CBASIC, but it only moves the 8K RS BASIC from ROM to RAM. The advantage is that you now have 39K for program storage.

CHECK

The CHECK utility is used to compare two disk files. The result 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 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.

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 that file's execution.

DESCRIPTION

CONTIN

Executing CONTIN will cause the message "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. An "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.

CONVERGE

CONVERGE is a type of convergence memory test, used primarily to detect address or data lines which are shorted together. If no parameters were specified memory from 0 to the FLEX "Memory end" is tested.

CONCAT

+++CONCAT,<file spec list>

This allows the listing and concatenation of several files either to the disk (If 0.CMD used) or the output device.

SUPER SLEUTH

A PROGRAM ANALYSIS AND DEBUGGING TOOL

by Edgar M. (Bud) Pass, Ph.D.

SUPER SLEUTH is a set of programs which enable the user to examine and/or modify object program files on disk or in memory on 6800 and 6809 systems running under FLEX*. Programs may be disassembled into source code format and the source may be displayed, printed, or saved on disk. Labels produced by SLEUTH can be changed globally to labels of the user's preference. Cross-reference listings of labels in any Motorola assembler-formatted source file may be produced to aid in debugging or modifying the program. Programs in ROM may be "altered" with the revised program being saved on disk; the resultant program could then be used to program a new ROM. Object code for 6800, 01, 02, 03, 05, 08, 09, or 6502 may be processed. 6800, 01, 02, 08, 09 object code may be easily converted to 6809 position-independent code.

SLEUTH (CSSLEUTH)

SLEUTH has been designed to run on either the 6800 or 6809 series of Motorola microprocessors under FLEX*. The object program it analyzes may reside on a FLEX* disk or in memory. The output files it produces may be sent to a FLEX* disk. SLEUTH is essentially self-instructive and has two levels of help files. It is a disassembler and object file editor for 6800, 01, 02, 03, 05, 08, 09, and 6502 CPU's.

SLEUTH has been designed for session-oriented use. A specific session normally begins when the program is executed, but may also begin when the 'R' (restart) command is issued by the user. All indicators are reset to their default values, as indicated below. The user may then issue various commands. All SLEUTH commands are capital letters. If any other input is found when a command is expected, the command menu is displayed. Because of the interdependence of many of the commands, they should be issued in the proper sequence and at the proper times. The explanation below is intended to serve as a quick guide to elementary operation of the SLEUTH.

SLEUTH is capable of processing 6800, 01, 02, 03, 05, 08, 09, and 6502 programs. By default, it assumes that the program is a 6800 program, unless SLEUTH has been assembled for and is running on a 6809 system, in which case it assumes that the program is a 6809 program. The 'Z' command may be used at any time to set the CPU-mode.

The user may need to determine the current indicator settings and control table contents. The 'L' command may be used at any time to list the indicators and tables.

If the user desires to process a program currently on disk, the 'S' command may be used. 'S' will prompt for an input file name. Since the 'S' command resets most indicators and tables, it should be issued only near the beginning of a session. If 'S' is not issued, or 'S' is issued but no file name is entered, then SLEUTH assumes that the object program resides in memory.

One command which should be used before an 'S' command, if desired, is the 'O' command. This command provides an offset value which is added to each address in the program being processed. If the program is to be processed from disk, the offset value is applied when the map is created (when the input file name is specified). If the program is to be processed from main memory, the offset value may be changed as often as desired, since the offset value is applied during the actual process of acquiring the data from memory.

If the program is being processed from disk, the starting and ending addresses will be set automatically. The transfer address will be set if it is present in the file. In any case, the 'N' command may be used to set or change the start, end and transfer addresses. The 'X' command may be used to set or change the transfer address. If the transfer address is set to FFFF, no transfer address will be generated in the output file. If an output is attempted and no start, end, or transfer address has been provided, they will be requested.

A label is normally placed on the line with the instruction or constant. If a label is to be placed other than at the beginning of a line of code, it is entered in one of the following formats:

```
label EQU *   or
label EQU *+n where n=1,2,3,4
```

The 'E' command may be used to flip a switch which will force those labels normally printed on the same line as the code to be printed in the format as specified above.

Normally, the disassembled code is generated such that the reassembled code will resemble as closely as possible the original object file. The 'B' command assists in producing such code by flagging all short address instructions with '<' and all long address instructions with '>'. The 'P' command specifies the generation of 6809 position- independent source code from 6800, 01, 02, 03, 08, or 09 object code. In the latter case, no particular attempt is made to produce code corresponding to the original object program.

At any point after start and end addresses have been defined, the 'Q' command may be used to format the program and display it on the terminal. Each page of the display shows 256 bytes of the program. The first page of the display begins at the address represented by the starting address with the low-order byte zeroed. Then the display may be paged forward, backward, set to an arbitrary 256-byte sector of memory, or terminated. A summary of the available commands will be displayed on the right edge of the screen. Each page of the display may also be modified in a full-screen edit manner. Data may be entered in either hexadecimal or alpha format, depending upon the area of the screen to which the cursor is pointing. The ASCII codes DC1, DC2, DC3 and DC4 (CTRL-Q, R, S and T) are used to move the cursor. Pressing CR (Return) will cause the editing mode to be exited and any changes that have been made will be registered. The displayed data represents the true resolved contents of the object program in main memory or on disk. If the input program file is composed of multiple redefinitions or the 'Q', 'M' or 'T' command has been used to change the value of the byte at a given address, only the last definition of a particular byte will be displayed.

The 'V' command is used to request a listing of the program code between the then-defined starting and ending addresses. This listing is produced in instruction, FCB, FCC, and FDB formats. The readability of the listed program code may be improved substantially in many cases thru the 'typing' of memory ranges. The 'Q' command may be used to help determine how to split memory into contigous ranges of instructions, constants, ignored areas, etc.

The 'D' command is used to perform a disassembly. If a start, end, or transfer address has not already been provided, they are requested. The user is requested for an output file name for the generated source program. If no file name is entered, no output disk file is created. A reassembleable source file (with labels) is generated and output as requested. All memory changes and memory types specified are honored. Thus, the interpretation of memory ranges as instructions may be overridden and FCB's, FDB's, FCC's may also be generated.

If the Flex "P" command is used preceeding the SLEUTH command (P CSSLEUTH) the user will be asked if output is to be sent to printer, in addition to display and disk, when the 'D' and 'V' commands are given.

Once an area of memory has been 'typed', the following commands may be used to improve the output display, printout, and disk source file:

```
A-FDB address range
C-FCC address range
H-FCB address range
```

I-instruction address range
J-instruction+ASCII address range
K-ignored address range

Each command above will request a memory range for the given type of memory. The last definition of a given byte is used in each case.

The 'M' command may be used to examine and change program code. When a 'M' is entered, a starting address is requested. The object code byte at this location is displayed. If the user desires to change the byte to another value, the new value may be entered. In this case, and in most other cases, the byte at the next location is displayed. If '↑' is entered, the byte at the previous location is displayed. If carriage-return is entered, the command is terminated. Since the specified memory location is not actually changed, object programs from any source (even disk or ROM) may be logically altered.

The 'T' command may be used to fill an entire range of program addresses with the same one-byte hex string. When this command is entered, the starting and ending addresses and one-byte hex string are requested.

The 'Y' command is used to scan for a hex string of bytes between a given range of addresses. The beginning addresses of the matching strings are printed in response. When the 'Y' command is entered, the starting and ending addresses and matching string are requested. A carriage return may be used to terminate the matching string.

The 'W' command is used to output the resulting object program to a disk file. If no output file name has been defined, one is requested. If the start, end and transfer addresses have not been provided, they are requested. The output file reflects only that program code between the start and end addresses, exclusive of ignored address ranges. Ignored address ranges may be generated explicitly (thru the 'K' command) or implicitly by not being defined in an input file. After the revised program code has been written to the output file, the transfer address is output if it is not equal to FFFF.

Although most of the SLEUTH commands are simple, repeated entry of a large number of them may become tedious and time-consuming, especially when a large program is being processed in an iterative fashion. The 'G' command allows the user to store commands such as 'A','C','H','I', 'J','K','T','M' in a text file and input them later to SLEUTH. It may be used whenever the '?' prompt is displayed. Any errors detected in the input text file cause the immediate termination of the reading of the file and return control to the terminal. The state of SLEUTH at a given time may thus be saved for a later execution.

The 'U' command may be used to execute a FLEX* command while still in SLEUTH. Any command which does not interfere with SLEUTH may be used. No check is made to determine whether a command is valid.

The 'F' command may be used to return to FLEX*.

NAME CHANGER (CSSNAMES)

The name-changer program is essentially a word substitution program. A table of words and substitutes is read into memory and the input file is read. All words in the input file are checked against the substitution table and, if a match is found, the appropriate change is made. The principal use of this facility lies in changing the machine-generated labels produced by SLEUTH into more mnemonic names.

The format of each record in the substitution file is as follows: /old-string/new-string/
where '/' may be any delimiter not in either string, new-string may be null, and old-string may not be null.

CROSS-REFERENCER (CSSXREFS)

The cross-reference program processes an assembly language source file and produces an alphabetically-sorted list of labels found in that file, the line number where each label is defined, and all line numbers on which that label appears. Any source program formatted according to the Motorola assembler source code format may be processed.

COPY

The COPY command is used for making copies of files on a disk. Individual files may be copied, groups of name-similar files may be copied, or entire disks may be copied. The copy command is a very versatile utility. The COPY command also re-groups the sectors of a file in case they were spread all over the old disk. This regrouping can make file access times much faster. It should be noted that before copying files to a new disk, the disk must be formatted first. Refer to NEWDISK for instructions on this procedure.

DESCRIPTION

The general syntax of the COPY command has three forms:

a. COPY,<file spec>,<file spec>
b. COPY,<file spec>,<drive>
c. COPY,<drive>,<drive>[,<match list>]

where <match list> is the same as that described in the CAT command and all rules apply to matching names and extensions. When copying files, if the destination disk already contains a file with the same name as the one being copied, the file name and the message, "FILE EXISTS DELETE ORIGINAL?" will be output to the terminal. Typing Y will cause the file on the destination disk to be deleted and the file from the source disk will be copied to the destination disk. Typing N will direct FLEX not to copy the file in question.

The first type of COPY allows copying a single file into another. The output file may be on a different drive but if on the same drive the file names must be different. It is always necessary to specify the extension of the input file but the output file's extension will default to that of the input's if none is specified. An example of this form of COPY is:

```
+++COPY,0.TEST.TXT,1.TEST25
```

This command line would cause the file TEST.TXT on drive 0 to be copied into a file called TEST25.TXT on drive 1. Note how the second file's extension defaulted to TXT, the extension of the input file.

The second type of COPY allows copying a file from one drive to another drive with the file keeping its original name. An example of this is:

```
+++COPY,0.LIST.CMD,1
```

Here the file named LIST.CMD on drive 0 would be copied to drive 1. It is again necessary to specify the file's extension in the file specification. This form of the command is more convenient than the previous form if the file is to retain its original name after the copying process.

The final form of COPY is the most versatile and the most powerful. It is possible to copy all files from one drive to another, or to copy only those files which match the match list characters given. Some examples will clarify its use:

```
+++COPY,0,1
+++COPY,1,0,.CMD,.SYS
+++COPY,0,1,A,B,CA.T
```

The first example will copy all files from drive 0 to drive 1 keeping the same names in the process. The second example will copy only those files on drive 1 whose extensions are CMD and SYS to drive 0. No other files will be copied. The last example will copy the files from drive 0 whose names start with 'A' or 'B' regardless of extension, and those files whose names start with the letters 'CA' and whose extensions start with 'T'.,to the output drive which is drive 1. The last form of copy is the most versatile because it will allow putting just the command (CMD) files on a new disk, or just the SYS files, etc., with a single command entry. During the COPY process, the name of the file which is currently being copied will be output to the terminal, as well as the drive to which it is being copied.

DATE

The DATE command is used to display or change an internal FLEX date register. This date register may be used by future programs and FLEX utilities.

DESCRIPTION

The general syntax of the DATE command is:

DATE[,<month,day,year>]

where 'month' is the numerical month, 'day' is the numerical day, and 'year' is the last two digits of the year.

+++DATE 9,5,83 Sets the date register to September 5, 1983.

Typing DATE followed by a carriage return will return the last entered date.

Example:

```
+++DATE
September 5, 1983
```

DBASIC for FHL Color FLEX

(Optional at a cost of $40.00)

DBASIC is a command for the Frank Hogg Laboratories implementation of FLEX for the Radio Shack TRS-80 Color Computer. It will not work with other versions of FLEX. It allows the use of the standard Disk Extended Color Basic under FLEX. All disk input and output operations are done through FLEX and are completely compatible with the normal FLEX utilities. This means that files and programs written to disk by DBASIC may be manipulated by FLEX editors, sort/merge, etc. It also means that these files are not compatible with standard Disk Color Basic files. However, the cassette files are compatible.

All of the BASIC language components described in the Radio Shack manuals are implemented, with the following exceptions:

1. Random files are not supported. Since you cannot open a random file, all of the commands that require a random file such as FIELD, LSET, RSET etc. will be of no use.

2. BACKUP, COPY, and DSKINI are not implemented and will give syntax errors. Use the equivalent FLEX utilities instead.

3. DIR is implemented differently. The output of DIR gives filename, extension, file size in sectors, and creation date. The two columns which give file type information under standard Color Basic are missing under DBASIC. That information is not stored in a FLEX directory. You must determine the file type by the extension. The file size is given in sectors, not granules.

4. FREE returns the number of free sectors, not granules.

5. DRIVE affects FLEX's default working drive. If you change it while in DBASIC, it will remain changed when you return to FLEX. Likewise, when DBASIC is started up, its default drive is the FLEX working drive.

6. VERIFY affects FLEX's verify flag. This should normally always be left ON.

7. LOC will return the current relative sector number of a file. This is not of great value for a sequential file.

8. LOF will return the file size of a file in sectors.

9. A new BASIC command called FLEX has been implemented. FLEX will terminate DBASIC and return to FLEX.

10. DSKI$ and DSKO$ are completely implemented. DSKO$ should not be used unless you a very familiar with the structure of a FLEX disk. You can crash a FLEX disk if you do not maintain the linkage bytes in every sector. See the PROGRAMMERS section of the FHL Color FLEX manual.

11. The disk driver entry point at $C004 (DSKCON) for machine language programs is not implemented. Calls to this routine will return with no action. Assembly Language subroutines called from DBASIC should not do disk I/O, even with calls to the FLEX FMS drivers. There is a conflict in the use of interrupts between BASIC and the FHL FLEX v5.0 disk drivers which will cause a system crash.

FILE NAMES AND DBASIC

DBASIC uses the same file name syntax as regular Radio Shack Disk Basic. This differs slightly from the way file names are specified under FLEX. In particular, under DBASIC the drive number, if specified, must be separated from the rest of the file name by a colon (:). Both DBASIC and regular R.S. Disk Basic accept either a period or a slash as the separator between the filename and extension.

Valid file specifications:

"PROGRAM"	Default drive and extension
"PROGRAM/BAS"	Default drive number.
"PROGRAM.BAS"	Period as separator is OK.
"1:PROGRAM.BAS"	Drive number specified.
"0:PROGRAM/BAS"	The way Radio Shack likes it.

Invalid file specification:

"1.PROGRAM.BAS"	DBASIC will not accept a period after the drive number.

FILE TYPES AND DBASIC

Standard Disk Extended Color Basic maintains two bytes in the directory entry of each file. The first byte is a number from 0 to 3 depending on whether the file is a BASIC program, BASIC data file, machine language file, or an editor source file. The second byte says whether the file is ASCII or Binary. Standard Disk Basic checks these file types whenever you use a file.

There are no equivalent bytes in the directory of a standard FLEX disk. Therefore, DBASIC neither checks nor sets the file types, and it is possible to LOAD a file which is not a program. You must be careful.

The default extension for a basic program is .BAS whether it is saved as a binary (tokenized) file or an ASCII file. An ASCII file is created by adding ",A" to the save command (see the SAVE command in your Color Computer Disk System manual). It is impossible to tell the difference from the directory unless you name them with unique extensions. A suggested method which is compatible in name only with other FLEX BASICS is to name the binary files .BAC.

The LOAD command determines whether a file is ASCII or binary by examining the first byte of the file, and will load either one. Binary (tokenized) BASIC programs load faster but cannot be processed by standard FLEX utilities.

INSTALLING DBASIC ON YOUR FLEX SYSTEM

DBASIC consists of two binary disk files, DBASIC.CMD and DBASIC.SYS. Both files must be present on the same disk to run. If you have purchased DBASIC along with FHL FLEX, both files should be on your system disk. If you have purchased it separately, copy both files to your system disk.

EXECUTING DBASIC

To execute DBASIC just type DBASIC at the +++ prompt from FLEX. If, for some reason you have DBASIC on another drive than your current system drive, you must of course specify the drive number in the standard FLEX fashion.

DBASIC may only be run from a non-32x16 Hi-res mode with MEMEND unchanged. The error message is "CANNOT BE RUN FROM THIS MODE."

HOW IT WORKS

DBASIC moves the code of the BASIC interpreter from ROM memory to the upper half of RAM, relocating some of it to avoid destroying FLEX. Only portions of the Disk Basic ROM are copied due to space limitations. It then reads in the changes necessary to run under FLEX in the new location. These changes are contained in the DBASIC.SYS file. DBASIC does not contain any of the original BASIC code which is copyrighted by TANDY and MICROSOFT.

SAVING DBASIC

Some of you may ask "How can I save the combination of the ROM code with DBASIC.SYS as a single binary file?" The answer is: You shouldn't and it won't do any good anyway. It would take longer to load than the present method. And, most importantly, it wouldn't work. There is no time when all of the code necessary for DBASIC to work exists in memory at the same time.

RTF - Radio Shack to FLEX copy program

RTF was written in DBASIC and will copy files from a Radio Shack disk to a FLEX disk. The program will work with single drive systems if you answer the prompts with the same drive number (0 in this case). In addition it will give a RS directory or a FLEX directory of a disk.

Although RTF will copy any RS disk file, a machine language file will be of little use when copied. This is because RTF does not modify the file to make it loadable with FLEX. There is a difference in the way RS and FLEX store ML code on the disk. RTF just does a byte for byte copy and this is the problem. Perhaps it is something better done after the file has been copied to a FLEX disk. However, all other files will copy and work ok, even BASIC BIN files.

```
RUN "0:RTF"
```

and then just answer the prompts.

Note: RTF will only copy ASCII files properly.

Also Note: LOADM requires a transfer address, even if not needed by the program. This is not a problem with files saved with SAVEM, but with files generated with an assembler.

DELETE

The DELETE command is used to delete a file from the disk. Its name will be removed from the directory and its sector space will be returned to the free space on the disk.

DESCRIPTION

The general syntax of the DELETE command is:

```
DELETE,<file spec>[,<file list>]
```

where <file list> can be an optional list of file specifications. It is necessary to include the extension on each file specified. As the DELETE command is executing it will prompt you with:

```
DELETE "FILE NAME"?
```

The entire file specification will be displayed, including the drive number. If you decide the file should be deleted, type 'Y'; otherwise, any other response will cause that file to remain on the disk. If a 'Y' was typed, the message 'ARE YOU SURE?' will be displayed on the terminal. If you are absolutely sure you want the file deleted from the disk, type another 'Y' and it will be gone. Any other character will leave the file intact. ONCE A FILE HAS BEEN DELETED, THERE IS NO WAY TO GET IT BACK! Be absolutely sure you have the right file before answering the prompt questions with Y's. Once the file is deleted, the space it had occupied on the disk is returned back to the list of free space for future use by other files. Few examples follow:

```
+++DELETE,MATHPACK.BIN
+++DELETE,1.TEST.TXT,0.AUGUST.TXT
```

The first example will DELETE the file named MATHPACK.BIN from the working drive. If auto drive searching is selected, the file will be deleted from the first drive it is found on. The second line will DELETE the file TEST.TXT from drive 1, and AUGUST.TXT from drive 0.

There are several restrictions on the DELETE command. First, a file that is delete or write protected may not be deleted without first removing the protection. Also a file which is currently in the print queue (see the PRINT command) can not be deleted using the DELETE command.

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 is listed with its file number, starting disk address in hex (track-sector), ending disk address, and file size in number of sectors. In addition, the file creation date and attributes are also displayed. Following the file name is an indication as to whether or not the file is a random file. 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 file's location in the directory, so the file numbers may not be consecutive if a lot of files have been deleted from the disk or a match list was specified. 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 begin with "A" and extensions begin with "T", as well as those files whose names start with "FR".

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 of 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 preceded 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 to the output device.

DYNAMIC

```
+++DYNAMIC[,<starting address>][,<memory size>]
```

DYNAMIC is used to test a block of 'dynamic' RAM for bit dropout under conditions such that the memory is not accessed for a period of time. Under these conditions, only the hardware refresh logic keeps the data current. 'Memory size' is the amount of memory to be tested in 1k blocks. If no parameters were given DYNAMIC will test 32 k starting at 0.

DYNAMITE

***DYNAMITE,input<file[,output<file][,+options[+command<file]]

Nice disassembler from CSC. Uses a predefined label file named DISLBL09 with standard FLEX definitions. The code and text boundaries in the file can be entered or provided in the command file.

Options:

A print ascii
B prompt for boundaries
D no output file
E empty line for label
F no default label file
G 1 line for FCC,FCB,FDB
L no listing
N print line numbers
P enable pagination
S prompt for segments
Z disassemble for 6800

command<file types:

+options: A,E,F,G,L,N
A FCC text
B FCB bytes
C instruction code
L FDB of labels
W FDB words
S label<file

Syntax for A,B,C,L,W:

<type> [<begin addr.>]-[<end addr.>]
<rpc>(<type>/<count>;<type>/<count>;..)

DUP

+++DUP,<drive #>,<drive #>
This lists the file names contained on one disk which are not on another one.

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 informative messages to the terminal to tell the operator of the status of the EXEC operation.

EXAMINE

+++EXAMINE[,<drive>]

This is the very nice interactive disk diagnostic from TSC.
The commands are:

```
R,<sector address>       Read a sector.
W,<sector address>       Write a sector.
D,<sector address>       Read and display a sector.
C,<sector address>       Read and display until end of file.
M,<byte number>          Modify sector number.
F,<file spec>            Read first sector of a file (1).
B,<file spec>            Build link table for a file (1).
T,<addr>,<addr>,<count>  Move data in memory (2).
S                        Stop, return to FLEX.
```

<sector address>=TTSS hex track,sector or + or - or =

1) system files: $B=boot, $S=SIR, $D=directory, $F=free chain
2) *B=256 byte buffer, *D=252 byte sector data, $1000-$B7FF free

EXTRACT

The EXTRACT utility is used to create a file from other files or segments of files. It is not necessary to copy the segments to scratch files and concatenate them. A command file is used to tell EXTRACT which lines are to be read from the files and concatenated to form the new file. Raw text may also be copied from the command file to the file being created.

DESCRIPTION

The general syntax of the EXTRACT command is:

EXTRACT,<new file>,<command file>

where <new file> is the specification of the file being created, and <directive file> is the name of a text file containing the directives. The <new file> must not already exist. The default extension for each of these files is TXT. EXTRACT reads the directive file, processing the directives in the order that they appear in the file, and creates the new file in accordance with the directives.

DIRECTIVES

A directive is a line in the directive file which starts with a right parenthesis, ")", in column 1. A line in the directive file which does not contain a ")" in column 1 is considered raw text and is immediately copied to the file being created. A directive has the following general form:

)<file spec><optional line list>

The <file spec> is a FLEX file specification. If no extension is specified, TXT is assumed. The file must, of course, already exist. The <optional line list> indicates which lines from the file are to be extracted and copied to the file being created. If no <optional line list> is specified, the entire file is copied. The <optional line list> consists of a series of single line numbers or line number ranges separated by commas. A line number range is a pair of line numbers separated by a hyphen (starting line-ending line). If the "starting line" is not specified, the beginning of the file is assumed. If the "ending line" is not specified, the end of the file is assumed. The line numbers and ranges in the line list do not have to be in ascending order; the file will be rewound, if necessary, in order to reach the line(s) being copied. If the last character on a directive line is a comma, the directive is assumed to continue starting in column 1 of the next line. Thus, directives may be continued across multiple lines. An

example follows:

```
EXTRACT NEW,INPUT
```

This command tells EXTRACT to create the file "NEW.TXT" from parts of the files mentioned on directives contained on the directive file "INPUT.TXT".

Assume that the directive file for the above example contains:

```
)FILEONE,5,7-10,2-3,50-
ADDITIONAL TEXT TO BE INSERTED
)FILETWO
)FILEONE,-10,15,20,
30,40-80
```

The file NEW will then contain, in order, lines 5, 7 through 10, 2 through 3, and 50 through the end of the file from the file FILEONE.TXT; the line ADDITIONAL TEXT TO BE INSERTED; all of FILETWO.TXT; and lines from the beginning of the file through line 10, lines 15, 20, 30, and 40 through 80 from the file FILEONE.TXT.

DISPLAY

The DISPLAY command is used to send strings of characters to the terminal or printer. These characters can be either hex, decimal or ascii.

DESCRIPTION

The general syntax of the DISPLAY command is:

```
DISPLAY <list>
```

where <list> is a list of characters or hex or decimal values to send to the terminal or printer.

```
DISPLAY $14,$20,$20        will send an Ctrl T and 2 spaces
DISPLAY 22,32,32           will do the same thing
DISPLAY "Hello"            will send the string <Hello>
DISPLAY $02,'=             will send 2 and an <=>
```

These can be intermixed on the same line:

```
DISPLAY $1B,'=,"AB","Hello There",27,'=
```

Hex characters must be preceded by a '$', single Ascii characters must be preceded by a "'", and strings of characters must be enclosed with '"'. Decimal numbers must not be preceded by anything. Spaces or commas must seperate characters. A error will be printed if a bad hex number is encountered and a syntax error for everything else.

This command can be used in the startup file to preset your terminal or if preceded by the 'P' command will send those characters to the printer. DISPLAY resides in the UCS and can be called from programs like Stylo to change the scroll rate of the screen.

EXAMPLES

```
DISPLAY 22,5                    Change to jump scroll
DISPLAY 22,1                    Change to smooth scroll

DISPLAY 2,$14,44,50,"MESSAGE"
```

Clear screen and print MESSAGE in the center of the screen.

```
DISPLAY 4,2,$14,56,40,"MESSAGE IN THE STATUS LINE",23,6
```

Kill status lines (4), clear screen (2), print the message on the 24th line ($14,56,40), set a status line and make it inverted (23 and 6).

EXEC

The EXECute command is used to process a text file as a list of commands, just as if they had been typed from the keyboard. This is a very powerful feature of FLEX for it allows very complex procedures to be built up as a command file. When it is desirable to run this procedure, it is only necessary to type EXEC followed by the name of the command file. Essentially all EXEC does is to replace the FLEX keyboard entry routine with a routine which reads a line from the command file each time the keyboard routine would have been called. The FLEX utilities have no idea that the line of input is coming from a file instead of the terminal.

DESCRIPTION

The general syntax of the EX command is:

```
EXEC,<file spec>
```

where <file spec> is the name of the command file. The default extension is TXT. An example will give some ideas on how EXEC can be used. One set of commands which might be performed quite often is the set to make a new system diskette on drive 1 (see NEWDISK). Normally it is necessary to use NEWDISK and then copy all .CMD and all .SYS files to the new disk. Finally the LINK must be performed. Rather than having to type this set of commands each time it was desired to produce a new system diskette, we could create a command file called MAKEDISK.TXT which contained the necessary commands. The BUILD utility should be used to create this file. The creation of this file might go as follows:

```
+++BUILD,MAKEDISK
  =NEWDISK,1
  =COPY,0,1,.CMD,.OV,.LOW,.SYS
  =LINK,1.FLEX
  =#
+++
```

The first line of the example tells FLEX we wish to BUILD a file called MAKEDISK (with the default extension of .TXT). Next, the three necessary command lines are typed in just as they would be typed into FLEX. The COPY command will copy all files with CMD, OV, LOW, and SYS extensions from drive 0 to drive 1. Finally the LINK will be performed. Now when we want to create a system disk we only need to type the following:

```
+++EXEC,MAKEDISK
```

We are assuming here that MAKEDISK resides on the same disk which contains the system commands. EXEC can also be used to execute the STARTUP file (see STARTUP).

There are many applications for the EXEC command. The one shown is certainly useful but experience and imagination will lead you to other useful applications.

IMPORTANT NOTE: The EXEC utility is loaded into the very upper end of user memory. This is done by first loading EXEC into the utility file space, then calculating the proper starting address so that it will reside right up against the end of the user memory space. Next EXEC is moved to that location and a new end of memory is set to just below EXEC. When the EXEC file is finished, if the user has not further changed the memory end location, EXEC will reset it to the original value.

EXT

External Terminal Utility

Ext will allow a standard serial terminal such as a TVI 910, to be hooked to the RS232 port of the Radio Shack Color Computer. Additionally, a printer may be hooked to the terminal. EXT uses the baud rate as set for the printer with SETUP. EXT also frees the 2K used by FLEX at the top of memory (B7FF-BFFF) by moving it to screen memory, and it too resides there.

This utility will control the capability built into the terminal that turns the terminal's printer port on and off.

DESCRIPTION

The general syntax of the EXT command is:

EXT,n,p1,p2,...pn,m,q1,q2,...qm,ec

where

n (decimal) is the number of characters in the sequence to disable printer pass-through on the terminal.

p1-pn (hex) are the characters in the sequence to disable printer pass-through. This sequence is also sent when EXT is run to initialize the terminal.

m (decimal) is the number of characters in the sequence to enable printer pass-through on the terminal.

q1-qm (hex) are the characters in the sequence to enable printer pass-through.

ec (hex) is the character generated by the input routine if a transition on the input line is detected by the status routine. If ec is 0, no character will be generated, although the true status will be returned.

A maximum of 12 characters may be supplied for each sequence.

The default calling sequence (for Televideo 910 terminals) is :

EXT,3,1B,61,0E,3,1B,60,1B

If any of the defaults are changed, all the parameters must be entered.

Note the "0E" added to the default disable sequence. This code disables the XON/XOFF software handshaking on the 910. The new EXT is sensitive to any transition on the input line for the purpose of interrupting printing (it is no longer necessary to use the BREAK key), and the XOFF generated when the buffer is full would interrupt output, requiring an ESCAPE to be input.

Also Note that EXT will not run if MEMEND is not B7FF. This condition applies if in a 32x16 mode, if already in EXT, or if MCOMMAND or something that changes MEMEND has been run. EXT prints the error message "CANNOT BE RUN FROM THIS MODE".

EXT

Extended Terminal Utility

INSTALLATION

The terminal is connected to the CC via the RS-232 port (serial I/O) on the back of the CC. This is a four connector DIN connector numbered 1,2,3 and 4. This is connected via cable to a DB25 connector. Pin 2, called "RD" and previously used for RS232 input, should now be connected to the DTR lead from the terminal. EXT uses this signal to implement true hardware handshaking.

Pin 1 of the DIN goes to Pin 2 of the DB25
Pin 2 of the DIN goes to Pin 20 of the DB25
Pin 3 of the DIN goes to Pin 7 of the DB25
Pin 4 of the DIN goes to Pin 3 of the DB25

The Microline 82A printer is connected to the terminal via a cable with two DB25 connectors.

Pin 1 of the 82A DB25 goes to Pin 1 of the 910
Pin 3 of the 82A DB25 goes to Pin 3 of the 910
Pin 7 of the 82A DB25 goes to Pin 7 of the 910
Pin 11 of the 82A DB25 goes to Pin 20 of the 910

The baud rate of the TVI 910 and the 82A are both set to 9600 baud. The SETUP command is used to set FLEX's baud rate at 9600 also. i.e: SETUP PB9600

Then the command EXT is executed and the '+++' will appear on the terminal. If you type 'P CAT 0' a catalog of drive zero should appear on the printer and the prompt should appear back on the terminal after the catalog is done.

The INT command is used to return to the Color Computer keyboard and display.

If your terminal cannot send characters to the printer without also sending them to the terminal (like the ADDS viewpoint), you can use the extra characters (up to 12) sent to the terminal after printing is done to reset the terminal (clear screen, set cursor, etc.) before continuing. This may need to be done if control codes are sent to the printer that may foul up the terminal.

There are many different hardware combinations (terminal/printer) that will probably work with EXT. But because of this great variety it is very hard to predict whether any particular set will work. We have included all the things in EXT that we could think of to make this job easier and will be very happy to modify and add to EXT whatever is necessary, if practical, to enhance it. If you have problems implementing EXT then send us, by mail, as much information about your problem as you can. Please don't call about this because you probably will not get someone that could help over the phone anyway. Things of this nature take time to consider and Ma Bell doesn't need your money that bad. Besides, you should be spending it on more software.

Good luck and enjoy this amazing and very useful program.

GET

The GET command is not a Disk Utility Command. It is a memory resident command (See page 1.7 in the FLEX User's Manual section). However, since many people expect it to be in the Utility Command section of this manual, we are including it here.

The GET command is used to load a binary file into memory. It is a special purpose command and is not often used.

DESCRIPTION

The general syntax of the GET command is:

GET[,<file name list>]

where <file name list> is: <file spec>[,<file spec>] etc.

The action of the GET command is to load the file or files specified in the list into memory for later use. If no extension is provided in the file spec, BIN is assumed. In other words, BIN is the default extension.

Examples:

+++GET,TEST
+++GET,1.TEST,TEST2.0

where the first example will load the file named 'TEST.BIN' from the assigned working drive, and the second example will load TEST.BIN from drive 1 and TEST2.BIN from drive 0.

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 of names 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'.

FILETEST

+++FILETEST[,<drive>][,<options>][,<file-list>]

FILETEST tests all files, or the file-list specified, for read errors. If no drive is specified the work drive is used.
Options:

A	Test all files
D	Test directory
F	Test free chain
M	Print sector list for each file
S	Test boot and system sectors

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 contain any printable (non-control) characters and is terminated by the carriage return or end of line character. Upon execution, all lines containing the specified string are printed on the terminal preceded by their line numbers. When finished, the total number of lines found containing the string is 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.

FIX

FIX is the interactive disk-as-memory binary file patch program.
The default extension is .BIN on the work drive.
The commands are:

```
  E               Exit, back to FLEX.
  L               List segments and transfer address.
  M addr.         Memory examine and change.

              ↑       previous address.
          <space>     next address.
            HH        change value into hex hh.
           <cr>       back to FIX prompt.

  N               Next 16 byte memory line.
  P addr.         Print 16 byte memory line.
  T               Set transfer address.
  U               Remove transfer address.
  V addr.-addr.   View memory in range.
  X               Abort.
```

FLAW

+++FLAW,[<drive>][,<sector list>]

FLAW removes sectors containing errors from the free chain of a diskette. Bad sectors detected by the user can be passed in the sector list and will be removed as well.
Run VALIDATE after FLAW to be sure nothing went wrong.

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.

FIX

FIX is the interactive disk-as-memory binary file patch program.
The default extension is .BIN on the work drive.
The commands are:

```
  E              Exit, back to FLEX.
  L              List segments and transfer address.
  M addr.        Memory examine and change.

            ↑       previous address.
         <space>    next address.
           HH       change value into hex hh.
          <cr>      back to FIX prompt.

  N              Next 16 byte memory line.
  P addr.        Print 16 byte memory line.
  T              Set transfer address.
  U              Remove transfer address.
  V addr.-addr.  View memory in range.
  X              Abort.
```

FLAW

+++FLAW,[<drive>][,<sector list>]

FLAW removes sectors containing errors from the free chain of a diskette. Bad sectors detected by the user can be passed in the sector list and will be removed as well.
Run VALIDATE after FLAW to be sure nothing went wrong.

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.

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 form feed (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 output an ASCII bell character (hex 7), then the string 'TEST', followed by another bell character.

HELPS

```
+++HELPS

+++HELPINST,<helpfile>
```

This is used to install the HELP.DAT help textfile. HELPINST reads the HELP.DAT file specified in 'helpfile' and creates a HELP.IDX index file on the same drive.

HELP

The HELP command is an online help utility of the type usually found on large computers. It reads text from a file called HELPCOCO.DIR. This file is a standard FLEX text file and as such can be added to or modified by any standard editor.

DESCRIPTION

The general syntax of the HELP command is:

HELP[,<search string><?>]

where <search string> is a string of characters (up to 8) in the same format as a FLEX command. HELP will look thru the file HELPCOCO.DIR for a match. HELP only looks at lines that begin with a printable character and when it finds a match it will print all lines that begin with a space, until it hits a line that begins with a non-space character, at which point it will stop and return to FLEX.

If <search string> ends with a <?> then all matches that begin with those characters will be printed.

EXAMPLE

HELP CAT

This will print information about the CAT command.

HELP D?

This will print information about anything that begins with a 'D', such as DIR, DUMP and DELETE.

Making changes to the HELPCOCO.DIR can be done with any FLEX editor. Remember that the first character in the line is importent. Look at the supplied file to understand the format.

You could, for example use this utility to find names in a name and address file, or parts in a part number file etc.

HELP for the TRS-80 color computer is a scaled down version of the more complete HELP utility written by Dale Puckett that Frank Hogg Laboratory, Inc. sells. The program is written in 6809 assembly language. ASM or ASMB is required for assembly. The larger HELP has such things as WILDCARD characters, optional file names, additional file names, and search string prompt. It would make a very fast mini database program. The cost is $29.95 for object only and $49.95 for the source included on the disk.

I

The I command allows a utility to obtain input characters from a disk file rather than the terminal.

DESCRIPTION

The general syntax of the I command is:

I,<file spec>,<command>

where <file spec> is the name of the file containing the characters to be used as input and <command> is the FLEX utility command that will be executed and that will receive that input from <file spec>. The default extension on <file spec> is .TXT.

For example, say that on a startup you always wanted the file DATA.DAT deleted from the disk without having to answer the "ARE YOU SURE?" questions. This could be done in the following manner:

```
+++BUILD,YES
=YY
=#
```

The first Y will answer the "DELETE 0.DATA.DAT?" question while the second Y will answer the "ARE YOU SURE?" question.

```
+++BUILD,STARTUP
=I,YES,DELETE,DATA.DAT
=#
```

Upon booting the disk, FLEX will execute the STARTUP file and perform the following operation: delete the file DATA.DAT receiving all answers to any questions from the input file YES.TXT rather than from the terminal.

See the description of the STARTUP command for more information on STARTUP.

INT

Return from EXT

INT is used after EXT to return to the Color Computer.

DESCRIPTION

The general syntax of the INT command is:

INT

INT will move code previously saved in high memory and reset FLEX vectors to point to the CC keyboard and screen.

INT will not run if MEMEND is not equal to BFFF. This will occur if MCOMMAND, etc. is run from EXT. The error message is "CANNOT BE RUN - MEMEND CHANGED".

FLEX VERSION 1
JANUARY 1983

PRODUCT WARRANTEE INFORMATION

The ISM (written by Frank Hoffman of LLOYD I/O) user manual, and object code software is supplied AS IS and without warrantee. Reasonable care has been taken to insure that the software does function as described in this manual. If you find a situation where the assembler does not function as the manual describes, then contact your dealer or LLOYD I/O. An attempt will be made to correct any errors brought to our attention, however we make no guarantee to do so.

Note: The following "full blown" assemblers from LLOYD I/O are available from your dealer. "ASM" is a standard 6809 macro assembler for FLEX and is available in a package with the "ED" text editor for $69.95, or alone for $50.00. "OSM" is a larger 6809 macro assembler available for OS9 or FLEX for $99.00. The cross assembler "CRASMB" is a larger assembler which allows cross assembling to other CPU's. It will handle programs for the 6800, 6801, 6805, 6809, 6502, 1802, 8080-8085, and Z80. It is available for FLEX or OS9 for $200 with a choice of one free CPU personality module (OS9 versions receive the 6809 module free). Each additional "CPM" (CPU Personality Module) is available for $35 (with source - $70).

Trademark Notices

FLEX is a trademark of Technical Systems Consultants, Inc., 111 Providence Road, Chapel Hill, NC 27514. OS9 is a trademark of Microware Systems Corporation, 5835 Grand Avenue, Box 4865, Des Moines, Iowa 50304.

INTRODUCTION

ISM is a interactive assembler. It assembles directly to memory. Therefore, take care when using it so you don't inadvertently erase parts of memory that must be left intact.

ISM will assemble any 6809 instruction using the standard Motorola syntax. You should use a guide to 6809 assembly language programming to help you learn and use the 6809 assembly language instruction set.

ISM fulfills two needs. One is to let you learn assembly language for the 6809 in a quick and responsive way. ISM assembles one line at a time just after you type it into the computer and then tells you if you made a mistake or not. Also it displays the machine code generated from the line just assembled. Second, ISM allows you to make quick changes to memory using the seven directives and the 6809 instruction set.

You may type the whole word "HELP" to get a list of the seven directives when you are using the assembler.

There are several things to keep in mind when reading this manual. Items enclosed within the greater-less than signs ('<' '>') are required, and items enclosed within brackets ('[' ']') are optional, that is they may be omitted.

INVOKING THE ASSEMBLER

The invoking systax for the assembler is:

ISM

This will cause the assembler to load and execute. Then typing 'A' will print a hexadecimal number followed by a space. The number is the current address where object code (bytes generated) will be stored. This number is referred to as the PROGRAM COUNTER. You may now type in any of the seven assembler directivesss or any 6809 instruction (mnemonic).

If you type in invalid mnemonics (directives or instructions) (pronounced "new-moniks") you will get error messages. Once the line is assembled, the code generated is displayed following the starting address of the code to the last byte of the code.

The next PROGRAM COUNTER position is displayed (in hexadecimal) and you may type in another mnemonic.

You must always refer to other memory locations by their actual address. Full blown assemblers allow you to define a memory location as a SYMBOL. Some programs can have hundreds or thousands of symbols. Using symbols makes it easy to write programs because the assembler can use the value of a symbol to automatically generate the right addresses for such things as jump and branch to subroutine instructions. In ISM you CANNOT use symbols. This is because it would require a larger assembler, symbol storage space, and two passes of all the lines of the program.

However ISM supports 12 "dummy" symbols which are some of the more commonly used FLEX systems calls. These are predefined and are a permanent part of ISM. Full blown assemblers allow you to define any symbol name you want, to any value you want with a maximum number of 30 characters for the LLOYD I/O assemblers.

This manual will contain definitions of the 12 symbols, the seven directives, and examples of how to use them.

ASSEMBLER DESCRIPTION

This assembler was written to accept standard assembler free format syntax. It follows then that the user must be familiar with assembly language format, and more particularly, with the Motorola format.

ISM accepts source code lines one at a time from the control port. The allowable characters are between $20 through $7E. Typing 'return' ends the input and allows the assembler to assemble the line you just typed. If you make a mistake while typing in instructions you may back-space by using your FLEX 'BACK-SPACE' character which is normally a control H (^H). If you want to delete the whole line you may type the FLEX 'DELETE' character which is normally a control X (^X).

Each line may be a maximum of 127 characters followed by a carriage return ($0D). Two fields are recognized by the assembler as valid code to process. These consist of (from left to right) the the OPERATOR (mnemonic), and the OPERAND. Fields are separated by one or more space characters ($20). Form:

<operator> [operand]

The restrictions and options for each field are as follows;

OPERATOR FIELD

1. The operator is one to six characters of the letters 'A' to 'Z', 'a' to 'z', and '0' to '9', and are followed by a space ($20).
2. Lower case letters are converted to upper case.
3. Mnemonics that use a register specifier may delete the seperating space.
4. Must start in the first position in the line.

OPERAND FIELD

1. All operands are generally considered an expression
2. The operands are evaluated for the expression value and the addressing mode used.
3. Some instructions do not require an operand so in that case, this field is omitted.

EXPRESSIONS

Expressions consist of combinations ofnumbers or labels separated by the operators. The arithmetic is done in 16 bit integers. Eight bit results are taken from the least significant 8 bits. Expressions must not contain spaces, and the expression is terminated when a space, carriage return, or an illegal character is found.

Expressions are evaluated according to the following list of operator precedence. More than one operator of the same type without parenthesis are evaluated left to right.

1. Parenthesized expressions
2. Unary plus and minus (+,-)
3. Shift operators (>>,<<)
4. Multiply and divide (*,/)
5. Addition and subtraction (+,-)
6. Relational operators (<,>,<=,>=,<>,=)
7. Logical NOT operator (!)
8. Logical AND and OR operators. (&,|)

NUMBERS

There are five types of numbers allowed. These are decimal (default), binary, octal, hexadecimal,and ASCII. This is a summary of the allowed formats for numbers:

BASE	PREFIX	POSTFIX	ALLOWED CHARACTERS
Decimal	none	none	0-9
Binary	%	not allowed	0-1
Octal	@	not allowed	0-7
Hexadecimal	$	not allowed	0-9,A-F,(a-f)
ASCII	'	not allowed	$20-$7F

Operators

type	operation	operator	form
math	add	+	value+value
	subtract	-	value-value
	multiply	*	value*value
	divide	/	value/value
logical	and	&	value&value
	or	!	value!value
	not	!	!value
	shift right	>>	value>>count
	shift left	<<	value<<count
relational	equal	=	value=value
	less than	<	value<value
	greater than	>	value>value
	less than or equal	<=	value<=value
	greater than or equal	>=	value>=value
	not equal	<>	value<>value

SYMBOLS

Symbols are names of values. They may be a symbolic value or the PC value. The first character must be in the ASCII range of (A-Z,a-z). The current PC value is defined by the character '*'. Upper case characters ARE equivalent to lower case characters.

SYMBOL EVALUATION

The following symbol names and their values are recognized by ISM. These are a permanent part of ISM and no others may be defined.

'*'	The current program counter's value
'WARMS'	$CD03 FLEX warm start entry point
'GETCHR'	$CD15 FLEX main input character subroutine
'PUTCHR'	$CD18 FLEX main output character subroutine
'INBUFF'	$CD1B FLEX input into line buffer
'PSTRNG'	$CD1E FLEX print a string of characters
'PCRLF'	$CD24 FLEX print carriage return, line feed
'NXTCHR'	$CD27 FLEX get the next character from input line
'OUTDEC'	$CD33 FLEX print a decimal number
'OUTHEX'	$CD3C FLEX print a byte as a hexadecimal number
'GETHEX'	$CD42 FLEX get a hex number from input buffer
'OUTADR'	$CD45 FLEX print an address as a hexadecimal number
'INDEC'	$CD48 FLEX get a decimal number from input buffer

OBJECT CODE FORMAT

This assembler generates object code which is stored directly into memory at the current PROGRAM COUNTER's value. Therefore take care when using ISM because it can erase parts of memory you had no intention of destroying.

ASSEMBLER DIRECTIVES

ISM supports the following directives or pseudo operators.

SUMMARY

FCC	form constant character(s)
FCB	form constant byte
FDB	form double byte
ORG	define a new origin (*)
END	signal end of assembly
RMB	reserve memory bytes
RUN	execute a program at a specified address

FCC

The function of FCC is to create character strings for messages, or tables. The character string 'text' is broken down to ASCII, with one character per byte. The format is:

label FCC delimiter text same delimiter

where the delimiter is used to define the starting and ending areas of the string. A maximum of 256 characters may be defined. There is another form for this directive. It is for example:

FCC "THIS IS TEXT",$0D,$0A,4

This variation allows strings to be set up for printing without having to use theform FCC,FCB. Commas may be used within the delimiters. There may be multiple strings as in this example:

FCC /This is text/,$0D,$0A,"This is another line",4

or

FCC /This, however is a line with a comma which is legal/,4

The only expressions recognized are decimal or hexadecimal, or an expression starting with a symbol.

FCB

FCB is used to evaluate an expression and use the results for an eight bit result. Multiple expressions are separated only by a comma, and may generate up to 256 bytes of object code. The format is:

label FCB expression 1,expression 2,...expression N

FDB

The directive FDB is the same as the FCB directive, however, 16 bit results are used rather than 8 bit.

ORG

The ORG directive causes a new origin address (PC) for the subsequent code that follows. The format is:

ORG expression

The default origin is zero (0000). This directive is used when you want to assemble code to a new location.

END

This directive causes the assembler to quit. The format is:

END

When you are done usirg the assembler this directive must be used to cause ISM to stop getting input and return to the monitor or FLEX.

RMB

This directive is used to 'Reserve Memory Bytes'. It is a relative originate directive. It simply adds the value of the expression which follows to the current PROGRAM COUNTER and creates a new value for the PROGRAM COUNTER. The format is:

RMB <expression>

RUN

This directive allows you to start a program running. It actually passes control to the address specifed. Format:

RUN <address>

Address is any legal expression. Control is passed to it as a subroutine. Therefore if you write a short program to do something it should end with a "RTS" (return from subroutine) or "JMP" to another location such as the FLEX "WARMS" address.

EXAMPLES

The following short program illustrates the use of the directives and some 6809 mnemonics.

```
0000 ORG $4000
4000 LEAX $4800,PCR
4004 JSR OUTADR
4007 LEAX $4802,PCR
400B JSR PSTRNG
400E RTS
400F ORG $4800
4800 FDB *+2
4802 FCC / THAT IS THE ADDRESS OF THIS STRING/
4825 FCB 4
4826 RMB 20
483A RUN $4000
483A END
```

Remember the mnemonic must start in the first column of each line. The assembler prints the current PROGRAM COUNTER value and one space.

ERROR MESSAGES

ISM supports the following error messeges.

1. UNDEFINED SYMBOL
2. ILLEGAL CHARACTER(S)
3. UNRECOGNIZABLE MNEMONIC
4. RELATIVE BRANCH TOO LONG
5. ILLEGAL ADDRESSING MODE
6. ILLEGAL REGISTER SPECIFIED
7. ABORTING..MEMORY OVERFLOW
8. ILLEGAL EXPRESSION
9. UNBALANCED PARENTHESIS IN EXPRESSION

The following mnemonics are supported to aid inconverting any 6800 programs to the 6809.

```
ABA  — STA B ,-S;ADD A ,S+
CBA  — STA B ,-S;CMP A ,S+
CLC  — ANDCC #$FE
CLF  — ANDCC #$BF
CLI  — ANDCC #$EF
CLV  — ANDCC #$FD
CLZ  — ANDCC #$FB
CPX  — CMPX
DES  — LEAS -1,S
DEX  — LEAX -1,X
INS  — LEAS 1,S
INX  — LEAX 1,X
SBA  — STA B ,-S;SUB A ,S+
SEC  — ORCC #$01
SEF  — ORCC #$40
SEI  — ORCC #$10
SEV  — ORCC #$02
SEZ  — ORCC #$04
TAB  — TFR A,B ; TSTA
TAP  — TFR A,CC
TBA  — TFR B,A ; TSTA
TPA  — TFR CC,A
TSX  — TFR S,X
TXS  — TFR X,S
WAI  — CWAI #$EF

SETDP — SET THE DIRECT PAGE VALUE AND FLAG

        SETDP <EXPRESSION> or SETDP ON or SETDP OFF
```

Examples of SETDP:

```
SETDP 0
SETDP $F0
SETDP LOW
```

The SETDP directive is used to allow the assembler to make the choice of whether or not to used direct page or extended addressing modes.

Extended and direct addressing modes may be forced by preceding the operand expression with an indicating character. Use the '>' greater than sign to force extended addressing and use the '<' less than sign to force direct addressing modes. As in:

```
STD >$00forces extended addressing
LDA <$EE23forces direct addressing
```

If neither mode is forced the assembler trys to use direct, by comparing the upper byte of the operand expression value to the current SETDP value. If they are not equal, or the SETDP is off, extended addressing modes are selected.

Some instructions allow for the form of:

STA A, STA B, STAA, STAB, STA, and STB. These include ST, OR, and LD. Also mnemonics that call for a register specification character(s) may be followed by a space before the register name. These types of mnemonics usually will have the register name printed where the normal position of acc. A or B was for the 6800 assembler.

Some instructions require an addressing mode which may be indexed. Most of these are indicated by the use of a comma (,)within the operand expression.For example "PCR" will not work for a 0,PCR value, itmust be ",PCR" or "0,PCR".

Auto increment-decrement is done as 0, X+ 0, X++, X+, X++ or 0, -X 0, —X , -X, —X.

The JUMP command is provided for convenience. It is used to start execution of a program already stored in computer RAM memory.

DESCRIPTION

The general syntax of the JUMP command is:

JUMP,<hex address>

where <hex address> is a 1 to 4 digit hex number representing the address where program execution should begin. The primary reason for using JUMP is if there is a long program in memory already and you do not wish to load it off of the disk again. Some time can be saved but you must be sure the program really exists before JUMPing to it!

As an example, suppose we had a BASIC interpreter in memory and it had a 'warm start' address of 103 hex. To start its execution from FLEX we type the following:

+++JUMP,103

The BASIC interpreter would then be executed. Again, remember that you must be absolutely sure the program you are JUMPing to is actually present in memory.

LINK

The LINK command is used to inform the boot loader ("FLEX/BAS") where the FLEX operating system resides on the disk. It is necessary to run LINK to make a disk usable for booting FLEX.

DESCRIPTION

The general syntax of the LINK command is

LINK,<file spec>

where <file spec> is usually FLEX. The default extension is SYS. LINK asks for confirmation of the command, allowing the disk to be changed if necessary.

An example of the operation of the LINK command follows:

```
+++LINK FLEX
LINK "0.FLEX.SYS"? Y
```


LIST

The LIST command is used to LIST the contents of text or BASIC files on the terminal. It is often desirable to examine a files without having to use an editor or other such program. The LIST utility allows examining entire files, or selected lines of the file. Line numbers may also be optionally printed with each line.

DESCRIPTION

The general syntax of the LIST command is:

LIST,<file spec>[,<line range>][,+(options)]

where the <file spec> designates the file to be LISTed (with a default extension of TXT),and <line range> is the first and last line number of the file which you wish to be displayed. All lines are output if no range specification is given. The LIST command supports two additional options. If a +N option is given, line numbers will be displayed with the listed file. If a +P option is given, the output will be formatted in pages and LIST will prompt for "TITLE" at which time a title for the output may be entered. The TITLE may be up to 40 characters long. This feature is useful for obtaining output on a printer for documentation purposes (see P command). Each page will consist of the title, date, page number, 54 lines of output and a hex 0C formfeed character. Entering a +NP will select both options. A few examples will clarify the syntax used:

```
+++LIST,RECEIPTS
+++LIST,CHAPTER1,30-200,+NP
+++LIST,LETTER,100
```

The first example will list the file named 'RECEIPTS.TXT' without line numbers. All lines will be output unless the 'escape character' is used as described in the Utility Command Set introduction. The second example will LIST the 30th line through the 200th line of the file named 'CHAPTER1.TXT' on the terminal. The hyphen ('-') is required as the range number separator. Line numbering and page formatting will be output because of the '+NP' option. The last example shows a special feature of the range specification. If only one number is stated, it will be interpretted as the first line to be displayed. All lines following that line will also be LISTed. The last example will LIST the lines from line 100 to the end of the file. No line numbers will be output since the 'N' was omitted.

MON

FHL Color FLEX contains a machine language monitor. The name of the monitor is 'MON'. Typing 'MON' will invoke the monitor while typing 'ROM' will return to RS Basic. If you are like me and have gotten used to typing 'MON' to return to RS Basic, then you will find the ^Q command useful. ^Q will return to RS Basic from 'MON'.

MON is a machine language monitor. The commands available are:

C.. Compare two blocks of memory.
D.. Display memory in HEX.
F.. Fill a block of memory with a value
J.. Jump to a user program (Break to exit).
M.. Memory examine and change
Space to advance - Backspace to go back.
^D.. Display memory in ASCII.
R.. Replace one byte with another within a range.
W.. Warm jump back to FLEX.
S.. Search a block for a byte.
T.. Transfer a block of memory.
X.. Register dump.
H.. Help list.
^X.. Change register.
^Q.. Quit and goto RS Basic.

The '^' means for you to hold the CTRL key (Shift and up arrow) while typing the character.

Remember that RS Basic is not in memory, so if you try to jump to it form MON you will probably go off into the ether, never to return, unless you hit the reset key - and that may not even work.

MOVEROM

The MOVEROM command is used to copy Radio Shack COLOR BASIC and Extended COLOR BASIC from ROM to RAM. Because of memory conflics, MOVEROM can only be used with an external terminal and the EXT utility.

DESCRIPTION

The general syntax of the MOVEROM command is

MOVEROM

This command will make an exact copy of the COLOR BASIC and Extended COLOR BASIC ROM's in RAM, at the exact same address. The user may then use other FLEX commands to study or modify the BASIC interpreter as desired.

N

The N (No) utility allows you to predetermine how a yes/no prompt from another utility will be answered.

DESCRIPTION

The general syntax of the N command is:

N,<COMMAND>

where the command is any applicable utility command file.

This command might be used when copying entire disks where some of the files being copied already exist on the destination disk and you don't wish them to be deleted. An example of this would be:

+++N,COPY 0,1

This will answer "NO" to a possible question of "FILE EXISTS. DELETE ORIGINAL?"

As you can probably see, you should exercise care in using this utility.

NEWDISK

The NEWDISK command is used to initialize a new or damaged disk for use with FLEX. NEWDISK is different from the Disk Extended COLOR BASIC DSKINI command, although the basic function performed is the same.

DESCRIPTION

The general syntax of the NEWDISK command is:

NEWDISK <drive number>

where <drive number> is the number of a disk drive into which the disk to be formatted will be placed. NEWDISK requests confirmation of the command from the user twice; if any answer but "Y" or "y" is given, NEWDISK will abort immediately.

NEWDISK then prompts the user for the following specifications: the number of sides, the density, and the number of tracks. If any reponse is incompatible with the disk drive (as indicated by the Drive Configuration Table), the command is aborted.

Finally, NEWDISK requests a Volume Name and Volume Number for the disk. At this time formatting of the disk begins.

NEWDISK may discover that the desired number of sectors (18 in double density, 10 in single) will not fit on the track. This is because NEWDISK uses a larger gap between the sectors than Disk Extended COLOR BASIC, for more reliable data storage. The exact track length varies from drive to drive. If the desired number of sectors will not fit, NEWDISK automatically reduces this number and reports "TRIMMING TRACK SIZE ...".

If NEWDISK does not trim the first track it formats but finds that trimming is required on a later track, formatting will be aborted. In this case the command will probably run correctly if issued again.

Note that all FLEX disks are single density on track 0.
An example of NEWDISK operation follows:

```
+++NEWDISK 0
ARE YOU SURE? Y
SCRATCH DISK IN DRIVE 0? Y
DOUBLE SIDED DISK? N
DOUBLE DENSITY DISK? Y
NUMBER OF TRACKS? 35
VOLUME NAME? MYDISK
VOLUME NUMBER? 42182
FORMATTING COMPLETE
TOTAL SECTORS = 612
```

NEWDISKA

NEWDISKA is the same as NEWDISK but uses a smaller gap between sectors and should be used with 40 or 80 track, single or double-sided drives. Using it will guarantee the full 18 sectors per track but could cause a problem if your drives are not up to snuff. Test it first. For the most part, 40 or 80 track drives are of a high enough quality that using NEWDISKA to format will not cause a problem.

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.

MEMEND

+++MEMEND[?][,<hex address>]

This is used to display or set the FLEX pointer to the top of user free memory. MEMEND alone will reset the pointer to its default value B7FF. Using the optional hex address you can set it to a desired value. MEMEND? will print ist current value.

MEMDUMP

+++MEMDUMP[,<start address>]

This will display one page of memory starting at start address. Then typing a 'F' will move to the next page and a 'B' will go to the previous page. A <CR> will return to FLEX.

MEMFILL

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

This fills memory from start to end with fill byte. If fill byte is left off the 0's are used.

MEMOVE

+++MEMOVE,<start address>,<endaddress>,<destination>

This will move memory from start to end address to destination address.

MEMTEST

+++MEMTEST,<hex start address>,<hexend address>

This is a memory test. You have to use reset to get out of it.

O

The O (not zero) command can be used to route all displayed output from a utility to an output file instead of the terminal. The function of O is similar to P (the printer command) except that output is stored in a file rather than being printed on the terminal or printer. Other TSC software may support this utility. Check the supplied software instructions for more details.

DESCRIPTION

The general syntax of the O command is:

O,<file spec>,<command>

where <command> can be any standard utility command line and <file spec> is the name of the desired output file. The default extension on <file spec> is .OUT. If O is used with multiple commands per line (using the 'end of line' character ':') it will only have affect on the command it immediately precedes. Some examples will clarify its use.

+++O,CAT,CAT
writes a listing of the current disk directory into a file called CAT.OUT

+++O,BAS,ASMB,BASIC.TXT
writes the assembled source listing of the text source file 'BASIC.TXT' into a file called 'BAS.OUT' when using the assembler

P

The P command is a memory resident command. It is very special, and unlike any of the commands currently in the UCS. P is the system print routine and will allow the output of any command to be routed to the printer. This is very useful for getting printed copies of the CATalog, or if used with the LIST command to print out copies of a FLEX text file.

DESCRIPTION

The general syntax of the P command is:

P,<command>

where <command> can be any standard utility command line. If P is used with multiple commands per line (using the 'end of line' character), it will only affect the command it immediately precedes. Some examples will clarify its use:

+++P,CAT
+++P,LIST,MONDAY:CAT,1

The first example would print a CATalog of the directory of the working drive on the printer. The second example will print a LISTing of the text file MONDAY.TXT and then display on the terminal a CATalog of drive 1 (this assumes the 'end of line' character is a ':'). Note how the P did not cause the 'CAT,1' to go to the printer. Comsult the 'Advanced Programmer's Guide' for details concerning the adaptation of the P command to various printers.

P1 and PRINT.SYS

FHL Color FLEX has a built in (memory-resident) printer driver as opposed to the standard disk-resident driver called 'PRINT.SYS' normally found with FLEX. Some standard FLEX programs may reference or call the PRINT.SYS file from disk by running the 'P' command (different than the one explained above - and included with FHL Color FLEX under the filename 'P1'.

For the sake of compatibility with FLEX programs that expect a PRINT.SYS file to reside on disk, a "dummy' PRINT.SYS file has been included on disk.

For more information on this, see page 1.9.

PCOPY

+++PCOPY

File copy program that works like COPY but prompts the user for each file asking permission to copy or not.
See COPY

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 along with a delete request:

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. If a response other than one the three above is given, the delete request will be posted again. 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!

PP

+++PP,<command>

Redirect output to the parallel interface printer.
Remember to use RM to reserve memory for the printer driver.
See P PRINTSYS RM S

PRINT

+++PRINT,<file spec>[,+<repeat>]

This command is used to put files in the print queue for Printer Spooling. The Printer Spooler is a background task in FLEX that prints files on the printer while the computer still accepts commands and surprisingly execOtes them as well. So there is no need to buy expensive large printer buffers since FLEX simply keeps its printer data in files on disk. Default extension is .OUT on the system drive.
See also O PSP QCHECK

The PROT command is used to change a protection code associated with each file. When a file is first saved, it has no protection associated with it thereby allowing the user to write to, rename, or delete the file. Delete or write protection can be added to a file by using the PROT command.

DESCRIPTION

The general syntax of the PROT command is:

PROT,<file spec>[,(option list)]

where the <file spec> designates the file to be protected and (option list) is any combination of the following options.

D A 'D' will delete protect a file. A delete protected file cannot be affected by using the DELETE or RENAME Commands, or by the delete functions of SAVE, APPEND, etc.

W A 'W' will write protect a file. A write protected file cannot be deleted, renamed or have any additional information written to it. Therefore a write protected file is automatically delete protected as well.

C A 'C' will Catalog protect a file. Any files with a C protection code will function as before but will not be displayed when a CAT command is issued.

X An 'X' will remove all protection options on a specific file.

Examples:

+++PROT CAT.CMD,XW	Remove any previous protection on the CAT.CMD Utility and write protect it.
+++PROT CAT.CMD,X	Remove all protection from the CAT.CMD utility.
+++PROT INFO.SYS,C	Prohibit INFO.SYS from being displayed in a catalog listing.

PUTBOOT.LDR

The PUTBOOT.LDR command is used to make a newly initialized disk usable to boot FLEX. The PUTBOOT.LDR command can not be copied from the original disk.

DESCRIPTION

The general syntax of the PUTBOOT.LDR command is

PUTBOOT.LDR <drive number>

where <drive number> is the number of a drive into which a newly formatted disk (see NEWDISK) will be placed. PUTBOOT.LDR asks for confirmation of the command (allowing the disk to be changed) after which it takes the necessary steps to place the Disk Extended COLOR BASIC file "FLEX/BAS" on the disk and prevent this file from being destroyed by FLEX.

If the disk is not newly formatted, or if it does not have 19 tracks, or if some critical sectors were found defective by NEWDISK, PUTBOOT.LDR aborts.

An example of the operation of PUTBOOT.LDR follows:

+++**PUTBOOT.LDR 0**
PUT BOOT LOADER ON DRIVE 0? Y

You can make as many bootable as you need. We suggest that you make several in case you damage your master. If you do damage the master you can have it replaced for only $10.00. Send your original disk along with proof of purchase and $10.00.

The QCHECK utility can be used to examine the contents of the print queue and to modify it contents. QCHECK has no additional arguments with it. Simply type QCHECK. QCHECK will stop any printing that is taking place and then display the current contents of the print queue as follows:

```
+++QCHECK
      POS       NAME      TYPE      RPT
       1        TEST.     .OUT       2
       2        CHPTR.    .OUT       0
       3        CHPTR2.   .TXT       0
  COMMAND?
```

This output says that TEST.OUT is the next file to be printed (or that it is in the process of being printed) and that 3 copies (1 plus a repeat of 2) of this file will be printed. After these three copies have been printed, CHPTR.OUT will be printed and then CHPTR2.TXT. The COMMAND? prompt means QCHECK is waiting for one of the following commands:

COMMAND	FUNCTION
(carriage return)	Re-start printing, return to the FLEX command mode.
Q	A Q command will print the queue contents again.
R,#N,X	An R command repeats the file at position #N X times. If X is omitted the repeat count will be cleared. Example: R,#3,5
D,#N	A D command removes the file at queue position #N. If N=1, the current print job will be terminated. Example: D,#3
T	A T command will terminate the current print job. This will cause the job currently printing to quit and printing of the next job to start. If the current files RPT count was not zero, it will print again until the repeat count is 0. To completely terminate the current job use use the D,#1 command.
N,#N	A N command will make the file at position #N the next one to be printed after the current print job is finished. Typing Q after this operation will show the new queue order. Example: N,#3
S	An S command will cause printing to stop. After the current job is finished, printing will hält until a G command is issued.

G	A G command will re-start printing after an S command has been used to stop it.
K	A K command will kill the current print process. All printing and queued jobs will be removed from the queue. The files are not deleted from disk.

P

QUICK

+++QUICK[,<starting address>,<ending address>]

The QUICK memory test performs a zeroes and ones check on a block of memory. This test is most frequently used as a quick check for solid failures. If no arguments were given, memory from 0 to FLEX "memory end" is tested.

RAWCOPY

+++RAWCOPY,<old-file>,<new-file>

RAWCOPY copies a file, ignoring CRC errors whenever possible. It is intended to be used in an attempt to retrieve most of the data in a file that has a bad sector in it. The default extension is .TXT on the work drive.

RANDOM

+++RANDOM[,<starting address>,<ending address>]

The RANDOM memory test tests a block of memory using pseudo-random bit patterns as the test data. If no arguments were given, memory from 0 to FLEX "memory end" is tested.

REBUILD

+++REBUILD,<source-drive>,<destination-drive>

REBUILD attempts to find files on a crashed diskette whose directory has been destroyed. Those files that are located are copied to another drive.

RECOVER

+++RECOVER,<source-drive>,<destination-drive>

RECOVER copies files from a crashed diskette to another diskette. Files to be copied are specified by their starting track and sector in hexadecimal numbers. The default extension of the filename is .TXT.

RENUMBER

+++RENUMBER,[<start line >],[<increment>]

This is called from BASIC and will renumber the program in memory the default are 10,10.

REMSPC

+++REMSPC,<input file spec>,<outputfile spec>

This is used to remove extra spaces from a file to save space. It will not change lines that start with a '*'.

REPLACE

+++REPLACE,<file spec1>,file spec2>

This will delete file spec 1 and rename file spec 2 file spec 1.

RPT

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

This will repeat any command any number of times.

RUN

The RUN command is used to load and optionally execute a position independent program at an address different from that at which the program normally executes.

DESCRIPTION

The general syntax of the RUN command is:

RUN,<load address>,<command> or
RUN/<load address>,<command>

where <load address> is that location at which the command is to be executed, and <command> is the command, with associated parameters, that is to be executed. The second form indicated above (with the slash following RUN) will cause <command> to be loaded at <load address>, but not executed. In this form, control will return to FLEX after the program is loaded. Be aware that this program does not change any of the instructions in the program being loaded. If the program is truly position independent, it will execute correctly at the new load address. Some examples follow.

+++RUN,1000,DEBUG
+++RUN/3500,TEST

The first example will load the program DEBUG.CMD at address 1000 hex and start it executing. The second example will load the program TEST.CMD at address 3500 but will not execute it. Control will be returned to FLEX instead.

RENAME

The RENAME command is used to give an existing file a new name in the directory. It is useful for changing the actual name as well as changing the extension type.

DESCRIPTION

The general syntax of the RENAME command is:

RENAME,<file spec 1>,<file spec 2>

where <file spec 1> is the name of the file you wish to RENAME and <file spec 2> is the new name you are assigning to it. The default extension for file spec 1 is TXT and the default drive is the working drive. If no extension is given on <file spec 2>, it defaults to that of <file spec 1>. No drive is requird on the second file name, and if one is given it is ignored. Some examples follow:

```
+++RENAME,TEST1.BIN,TEST2
+++RENAME,1.LETTER,REPLY
+++RENAME,0.FIND.BIN,FIND.CMD
```

The first example will RENAME TEST1.BIN to TEST2.BIN. The next example RENAMEs the file LETTER.TXT on drive 1 to REPLY.TXT. The last line would cause the file FIND.BIN on drive 0 to be renamed FIND.CMD. This is useful for making binary files created by an assembler into command files (changing the extension from BIN to CMD). If you try to give a file a name which already exists in the directory, the message:

FILE EXISTS

will be displayed on the terminal. Keep in mind that RENAME only changes the file's name and in no way changes the actual file's contents.

One last note of interest. Since utility commands are just like any other file, it is possible to rename them also. If you would prefer some of the command names to be shorter, or different all together, simply use RENAME and assign them the names you desire.

ROM

ROM is a memory resident as opposed to disk resident command. It is included in this section of the manual since most poeple tend to look for it here.

Rom is used to exit FLEX and return to Radio Shack Disk Extended BASIC.

DESCRIPTION

The general syntax of the ROM command is:

ROM

followed by the 'ENTER' key.

NOTE: To re-enter FLEX after using the ROM command, you have to re-boot FLEX. If you wish to use Radio Shack Basic and be able to return to FLEX, then use either CBASIC, BASIC or the optional DBASIC.

R

S

+++S,<command>

Redirect output to the serial interface printer.
Remember to use RM to reserve memory for the printer driver.
See P PP PRINTSYS RM

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 at which 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.

SAVE

The SAVE command is used for saving a section of memory on the disk. Its primary use is for saving programs which have been loaded into memory from tape or by hand.

DESCRIPTION

The general syntax of the SAVE command is:

SAVE,<file spec>,<begin adr>,<end adr>[,<transfer adr>]

where <file spec> is the name to be assigned to the file. The default extension is BIN and the default drive is the working drive. The address fields define the beginning and ending addresses of the section of memory to be written on the disk. The addresses should be expressed as hex numbers. The optional <transfer address> would be included if the program is to be loaded and executed by FLEX. This address tells FLEX where execution should begin. Some examples will clarify the use of SAVE:

```
+++SAVE,DATA,100,1FF
+++SAVE,1.GAME,0,1680,100
```

The first line would SAVE the memory locations 100 to 1FF hex on the disk in a file called DATA.BIN. The file would be put on the working drive and no transfer address would be assigned. The second example would cause the contents of memory locations 0 through 1680 to be SAVEd on the disk in file GAME.BIN on drive 1. Since a transfer address of 100 was specified as a parameter, typing 'GAME.BIN' in response to the FLEX prompt after saving would cause the file to be loaded back into memory and execution started at location 100.

If an attempt is made to save a program under a file name that already exists, the prompt "MAY THE EXISTING FILE BE DELETED?" will be displayed. A Y response will replace the file with the new data to be saved while a N response will terminate the save operation.

Sometimes it is desirable to save noncontiguous segments of memory. To do this it would be necessary to first SAVE each segment as a separate file and then use the APPEND command to combine them into one file. If the final file is to have a transfer address, you should assign it to one of the segments as it is being saved. After the APPEND operation, the final file will retain that transfer address.

SAVE.LOW

There is another form of the SAVE command resident in the UCS. It is called SAVE.LOW and loads in a lower section of memory than the standard SAVE command. Its use is for saving programs in the Utility Command Space where SAVE.CMD is loaded. Those interested in creating their own utility commands should consult the 'Advanced Programmer's Guide' for further details.

SDC

The Single Drive Copy command is used to copy files from one disk to another, when only a single disk drive is available.

DESCRIPTION

The general syntax of the SDC command is

SDC,<file spec>[,<file list>]

where <file list> is <file spec>[,<file spec>] etc.

For each file, SDC will ask the user to insert the source disk (the disk containing the file to be copied) into the drive, after which as much of the file will be copied into memory as possible. Then the user is asked to insert the destination disk (to which the file is to be copied) into the drive. This process is repeated, if necessary, until the entire file is copied, after which SDC reports FILE COPIED. The entire process is repeated for each file, and then the message SDC COMPLETE appears and the command terminates.

The file must not already exist on the destination drive, or SDC will abort.

An example of SDC operation follows:

```
+++SDC 0.FLEX.SYS
*INSERT SOURCE DISK THEN HIT KEY
 INSERT DESTINATION DISK THEN HIT A KEY
 FILE COPIED
 SDC COMPLETE
```

* The 'SOURCE DISK' is the one that the file you want to copy is on, while the 'DESTINATION DISK' is the disk where you want the file to go to.

SETUP

The SETUP command is used to set up various options within the FLEX operating system, or to generate a binary file to be appended to FLEX.SYS which changes the setting of various options.

DESCRIPTION

The general syntax of the SETUP command is:

SETUP <option field> [<option field> ...]

where <option field> = <key character><options>. The option field must not contain spaces, and the option fields must be separated by spaces. The key character determines how the options in each option field are interpreted.

The descriptions of option fields for the terminal (key character=T), for the printer (key character=P), for memory (key character=M), and for the disk system (key character=0,1,2, or 3) are described on separate pages. The option field for binary file generation (key character=F) is described below. The following basic syntax rules apply:

1. All alphabetic characters must be UPPER CASE.
2. Only valid characters (no extra separators or spaces) may appear in option fields.
3. Spaces must separate option fields.
4. <dec> is a decimal number, which must be terminated by a comma.
5. <hex> is a hexadecimal number, which must be terminated by a comma.

Rules 4 and 5 are very important. The number scanning routine (internal to FLEX) used by SETUP will skip over alphabetic characters that come after a number, if the number is not terminated. Undesired results will occur if the comma is left out.

For example, the string

PB110S3, (see SETUP-PRINTER)

would be interpreted as

PB1103,

which is not as desired.

Also note that numeric characters serve as key characters or option characters in some places. These characters are given explicitly in the SETUP documentation, and must NOT be terminated by a comma. Only numbers given as <dec> or <hex> in the documentation get terminators.

SETUP will parse all options before taking action. If an error is detected, none of the options specified will be implemented. Instead, SETUP will reprint the command line up to and including the character which caused the error, indicating "ERROR ENCOUNTERED AT THIS POINT."

Normally, SETUP will implement the options in memory before terminating. If the F key character is used, the options will be placed in a binary file instead of in memory. The F option field must be the first option field if used. the syntax of the F option field is:

F<filespec>

The file must not already exist.
The file extension defaults to BIN.

Some examples of the use of SETUP follow. (See the sections on the specific option fields of SETUP.)

Make the screen green, make the cursor a blinking block and install drive one which is a double-sided 40 track drive with a 6 millisecond stepping rate:

```
+++SETUP TGCC 1DT40,6
```

Make a new FLEX.SYS on drive 1 which contains the above options:

```
+++SETUP F1.FLEXMODS.BIN TGCC 1DT40,6
+++APPEND 0.FLEX.SYS 1.FLEXMODS.BIN 1.FLEX.SYS
```


SETUP DISK

The disk option field of the SETUP command is used to change parameters in the Drive Configuration Table, which FLEX uses to control disk operations. The Drive Configuration Table should reflect the actual drives wired into the system.

DESCRIPTION

The general syntax of the SETUP command with the drive option is:

SETUP [<other option fields>] <drive number><drive options> [<other option fields>]

where <drive number>=0-3 and <drive options> are described below. See the section on SETUP for more information.

Using SETUP with a drive option will cause all Configuaration Table entries for that drive to be reset to the defaults, except for entries mentioned in the option field.

The drive options are as follows:

OPTION	DESCRIPTION	DEFAULT
D	Double-sided (two-head) drive	**double** *
S	Single-density drive	**double**
6	6ms track-to-track step rate	**no**
1	12ms track-to-track step rate	**no**
2	20ms track-to-track step rate	**no**
3	30ms track-to-track step rate	**yes**
T<dec>,	Set number of tracks drive can access <dec>=1-255	**40**
W<dec>	begin write precompensation at track <dec>	**22**
p<dec>	Set physical (hardware) drive number of this drive to <dec> (<dec>=0-3).	**same as logical**

(This option allows FLEX references to a drive to actually access a different drive. Note that FLEX must be booted in physical drive 0.)

CAUTION: No two logical drive numbers should share the same physical drive number.

Examples:
Add 3 Radio Shack Drives, 1,2 and 3: +++SETUP 1T35,2T35,3T35

Add 3 40 Track Drives, 1,2 and 3: +++SETUP 1 2 3

Swap references to drives 2 and 3:

```
+++SETUP 2P3, 3P2,
```

Make drive 3 a double-sided, 80-track, 6ms-step drive:

```
+++SETUP 3DT80,6P2,
```

Note that the P2 was repeated, otherwise it would have defaulted back to P3.

* 'SETUP 0' will set up your drive as single sided.

SETUP MEMORY

The Memory option field of the SETUP command may be used to examine or change memory locations. It is provided with FHL COLOR FLEX because the TRS-80 COLOR COMPUTER does not have a built-in monitor program.

DESCRIPTION

The general syntax of the SETUP command with the M option is:

SETUP [<other option fields>] **M**<memory options> [<other option fields>]

where <memory options> are described below. See the section on SETUP for more information.

No separating characters of any kind may be typed between the options.

The **M** options are as follows:

OPTION	DESCRIPTION
E<hex>,	Examine location <hex>

NOTE: The use of the F option (see SETUP) has no effect on the ME option.

S<hex>,<hex>, Set location <hex1> to <hex2>

Examples:

```
+++SETUP ME1A0,E1A1,
01A0 4B
01A1 6D
+++SETUP MS1A0,2C,E1A0,
01A0 4B
+++SETUP FJUNK.BIN ME1A0,S1A1,35,
01A0 2C
+++SETUP ME1A1,
01A1 6D
+++GET JUNK.BIN
+++SETUP ME1A1,
01A1 35
```

Note that all Examines are done before any Sets, regardless of the order in which they are inputted. Also note that the F option works for Set but not for Examine.

SETUP PRINTER

The Printer option field of the SETUP command is used to change certain parameters in the FLEX printer output software.

DESCRIPTION

The general syntax of the SETUP command with the P option is:

SETUP [<other option fields>] **P**<printer options> [<other option fields>]

where <printer options> are described below. See the section on SETUP for more information.

The printer options have certain defaults which are the values of the options in the FLEX.SYS file on the original FHL COLOR FLEX disk. Using the P option field does not reset any options to the defaults, unless the option is mentioned specifically in the field.

No separating characters of any kind may be typed between the options.

The P options are as follows:

OPTION	DESCRIPTION	DEFAULT
B<dec>,	Set baud rate (110-9600)	**600**

NOTE: The SETUP command chooses the value to give the best possible approximation of any baud rate in the software. Since the baud rate cannot be set exactly, some printers may not function correctly when the baud rate is initially set. If errors are observed on the printer, try adjusting the baud rate by +/-6%. Also try adding more stop bits (see below).

N	Normal (sends line feeds through)	**no**
R	Radio Shack (assumes auto-lf)	**yes**

NOTE: When using Radio Shack-style printers, output carriage return- line feed as you would to a normal printer. With SETUP PR, the printer output routine translates cr-lf to cr, and translates extra lf's to cr's.

S<dec>, Set stop bit count (1-255) Default=**1**

For example, to setup for a 2400-baud, normal printer that likes its data a little slow:

```
+++SETUP PB2200,N
```

To cut the data transfer rate in half without changing the baud rate:

```
+++SETUP PS10,
```

SETUP TERMINAL

The Terminal Option Field of the SETUP command is used to change certain parameters in the FLEX keyboard input/output software.

DESCRIPTION

The general syntax of the SETUP command with the T option is:

SETUP [<other option fields>] T<terminal options> [<other option fields>]

where <terminal options> are described below. See the section on SETUP for more information on the other option fields.

The terminal options have certain defaults which are the values of the options in the FLEX.SYS file on the original FHL COLOR FLEX disk. Using the T option does not reset any options to the defaults, unless the option is mentioned specifically in the field.

NOTE: Do not type any separating characters between the options.

The T options are as follows:

OPTION	DEFAULT	DESCRIPTION
B	40	Set time between cursor blinks (relative scale)
CC	NO	Blinking block cursor
CI	NO	Non-blinking underline cursor
D<dec>,	7	Set number of key scans for debouncing
F<dec>,	300	Set pitch of Ctrl-G tone (higher number yields lower tone
G	NO	Set terminal screen to green background
L<dec>,	30	Set length of Ctrl-G tone (it is also dependant on the pit
M<dec>,	20	Set number of blinks before motors turn off (even if a non blinking cursor is used, the blinks are still counted at the rate and used as a basis for turning off the motors.)
W	YES	Set terminal screen to white background

STARTUP

STARTUP is not a utility command but is a feature of FLEX. It is often desirable to have the operating system do some special action or actions upon initialization of the system (during the bootstrap loading process). As an example, the user may always want to use BASIC immediately following the boot process. STARTUP will allow for this without the necessity of calling the BASIC interpreter each time.

DESCRIPTION

FLEX always checks the disk's directory immediately following the system initialization for a file called STARTUP.TXT. If none is found, the three plus sign prompt is output and the system is ready to accept user's commands. If a STARTUP file is present, it is read and interpreted as a single command line and the appropriate actions are performed. As an example, suppose we wanted FLEX to execute BASIC each time the system was booted. First it is necessary to create the STARTUP file:

```
+++BUILD,STARTUP
  =BASIC
  =#
+++
```

The above procedure using the BUILD command will create the desired file. Note that the file consisted of one line (which is all FLEX reads from the STARTUP file anyway). This line will tell FLEX to load and execute BASIC. Now each time this disk is used to boot the operating system, BASIC will also be loaded and run. Note that this example assumes two things. First, the disk must contain FLEX.SYS and must have been LINKed in order for the boot to work properly. Second, it is assumed that a file called BASIC.CMD actually exists on the disk.

Another example of the use of STARTUP is to set system environment paramters such as TTYSET parameters or the assigning of a system and working drive. If the STARTUP command consisted of the following line:

```
TTYSET,DP=16,WD=60:ASN,W=1:ASN:CAT,0
```

each time the system was booted the following actions would occur. First, TTYSET would set the 'depth' to 16 and the 'width' to 60. Next, assuming the 'end of line' character is the ':', the ASN command would assign the working drive to drive 1. Next ASN would display the assigned system and working drives on the terminal. Finally, a CATalog of the files on drive 0 would be displayed. For details of the actions of the individual commands, refer to their descriptions elsewhere in this manual.

As it stands, it looks as if the STARTUP feature is limited to the execution of a single command line. This is true but there is a way around the restriction, the EXEC command. If a longer list of operations is desired than will fit on one line, simply create a command

file containing all of the commands desired. Then create the STARTUP file placing the single line:

EXEC,<file name>

where <file name> would be replaced by the name assigned to the command file created. A little imagination and experience will show many uses for the STARTUP feature.

By directing STARTUP to a file that does not have a return to DOS command it is possible to lockout access to DOS. You can correct the problem by hitting the RESET button and beginning execution at address $CD03. The STARTUP file may then be deleted and if desired, modified. Directing execution to CD03, the DOS warm start address, bypasses the DOS STARTUP function.

TED

Tiny Editor

TED is a editor for FLEX. It works with standard FLEX text (.TXT) files. The only limitations are that of 255 lines and 128 characters per line.

TED edits and/or creates files that can be used by any FLEX program that needs standard ASCII files. TEDs files are compatible with all other known editors available for FLEX.

The syntax for TED is:

+++**TED**

TED will prompt for the filename you wish to edit. The default drive is the working drive and the default extension is '.TXT'. If you have set drive 1 as the working drive, then an answer of '1.TEST.TXT' for the filename would be the same as answering 'TEST'.

TED will then try to read in a file by the name you have given it. If the file is too large to fit into TED's buffer then TED will tell you that and exit back to FLEX.

If the file does fit then TED will read it in to its buffer and display the menu. If the file does not exist, (a new edit) then TED will just display the menu and put you into Command mode. Command mode is when the TED prompt 'TED:' is displayed. This means that TED is waiting for you to tell it what to do from one of the menu selections below.

Here is what the menu will look like;

```
          L - List the file in memory.
          S - Save the file in memory to disk.
          F - Find a string of characters in the file.
          D - Delete the current line.
          E - Edit the current line.
          I - Insert after the current line.
          N - New, erase all lines in memory.
          G - Goto a line number.
      ENTER - Display current line.
   UP ARROW - Goto line 1
   DN ARROW - Goto last line
   LF ARROW - Backup one line.
   RT ARROW - Forward one line.
          H - Display this menu
```

In addition to the above menu commands, there is also:

A - Abort the file
P - Print (n) lines
CONTROL X (shift, up arrow X) - which will cancel any string being entered.

Some of the above are obvious, while others need some explanation. Here is a rundown of what these commands do. I suggest that you play with TED for awhile before you try to use it for anything serious.

List

List (L) will prompt for the range or lines to list. It will only list as many lines as are in the file. The maxium number of lines is 255.

SAVE

Save (S) the file in memory to disk under the name you had typed in the beginning. If the file was on the disk then TED will ask if you wish to save this file under a different name. If you answer no (N) then TED will delete the file on the disk and then save the file in memory and return to FLEX. You can also exit TED without saving anything to disk by aborting. This last is useful for learning how TED works.

FIND

Find (F) a string of characters. Find will prompt for the string to search for. Type it in, ending it with a (ENTER). Find will stop at the first occurance of the string and make the line that the string was in, the current line. Find will display the line then return to the command mode. Then, just typing an **F** and hitting enter again will cause TED to find the first occurance of that string in successive lines.

DELETE

Delete (D) will delete the current line. Caution, delete does not prompt you to verify that this is what you really want to do. Typing a (D) will delete the line!

PRINT

Print (P) lines. Typing **P** in the command mode will cause TED to ask you how many lines of your file you would like to have printed on your screen. Just hitting enter will print out the next ten lines of your file.

ABORT

Abort (A) a file. Typing A while in the command mode will cause the current file to be erased from memory and control returned to FLEX.
Note: The same result may be achieved by using Save (S) which will ask if you want to save to disk or abort.

EDIT

Edit (E) will put you into the edit mode for the current line. Edit will display the current line and then print the line number and wait for your input. At this point you have several options.

1.. You can type in any characters you wish. They will be inserted **before** the character in the line.

2.. You can type a **UP ARROW** character followed by any other printable character and the line will be printed to the first occurance of that character.
Note: You may use either the BREAK key or the up arrow interchangeably with TED. This is a noteworthy feature if you're using the EXT command with FLEX since the up arrow on a terminal is generated by a Control K and could prove to be a bit of a pain in the neck to use.

3.. You can type a left arrow, which will delete the character to the left of the cursor.

Using the above three commands will allow you to move very quickly thru the line to get to the place you want to edit. If for instance you wanted to get to the **'p'** in the word 'pool' below. You would type 'BREAK' then **'p'** which would get you to the **'p'** in **p**lace, then 'BREAK' and **'p'** again would get you to the **'p'** in 'pool'.

'Pick the **p**lace where you want the **p**ool to go.'

Here are the rest of the commands and their uses:

4.. If you type a ENTER, you will exit the edit mode with all changes made.

5.. If you type a DOWN arrow, then the line will be chopped off at the cursor and you will exit the edit mode with all changes made.

6.. If you type a RIGHT arrow you will got to the end of the line without exiting the edit mode. This is useful for adding things to the end of the line.

INSERT

Insert (I) AFTER the current line. This will put you into insert mode. Type a # in the first column to exit the insert mode.

If the current line number is 23 then the line number will be 24...on in the insert mode. Line numbers are internal to TED and are only used for reference. Insert will not delete any lines, they will only have their internal line numbers changed. The line limit is 128 characters and TED will truncate anything after that.

NEW

New (N) will erase all lines in memory. It is similar to the NEW command of Basic.

GOTO

Goto (G) a line number. Goto will prompt for the line number to go to. Goto will make that line the current line.

ENTER

Typing the ENTER key in command mode will display the current line.

Up arrow

Typing an up arrow will make line 1 the current line and put you at the beginning or top of the file.

Down arrow

Typing an down arrow will make the last line of text the current line and put you at the end or bottom of the file.

Left arrow

Typing a left arrow will back up one line and make that the current line.

Right arrow

Typing a right arrow will advance one line and make that the current line.

H

Typing an 'H' will display the menu again.

Well that's it. TED is a small but powerful editor that can be put to good use for FLEX. TED was not meant to replace a full size editor like FHL ED or FHL DynaStar ect. However TED is indeed a very useful editor and will serve you well.

For the curious.

TED was written using the A/BASIC compiler from FHL. The source listing is only four pages long. Text is stored in memory in a string array 255 X 128.

I gave some thought to making TED more powerful by adding several features. However as usual I was getting carried away. We created TED so that new users of FLEX would not have to buy a full blown editor like ED.

Several of these new users had mentioned that their use of an editor was only ocassional and they could't justify the cost of one for that use. TED was created to serve that need. A small, easy to use editor for the ocassional user.

I've had a lot of fun writing TED, and I hope that you will have as much fun using it.

If you have any suggestions for improving TED please send them in. However, remember that TED is not meant to replace a full editor or for that matter, to become one.

Thank You

Frank Hogg

TTYSET

The TTYSET utility command is provided so the user may control the characteristics of the terminal. With this command, the action of the terminal on input and the display format on output may be controlled.

DESCRIPTION

The general syntax of the TTYSET command is:

```
TTYSET[,<parameter list>]
```

where <parameter list> is a list of 2 letter parameter names, each followed by an equals sign ('='), and then by the value being assigned. Each parameter should be separated by a comma or a space. If no parameters are given, the values of all of the TTYSET parameters will be displayed on the terminal.

The default number base for numerical values is the base most appropriate to the parameter. In the descriptions that follow, 'hh' is used for parameters whose default base is hex; 'dd' is used for those whose default base is decimal. Values which should be expressed in hex are displayed in the TTYSET parameter listing preceded by a '$'. Some examples follow:

```
+++TTYSET
+++TTYSET,DP=16,WD=63
+++TTYSET,BS=8,ES=3
```

The first example simply lists the current values of all TTYSET parameters on the terminal. The next line sets the depth 'DP' to 16 lines and the terminal width, 'WD' to 63 columns. The last example sets the backspace character to the value of hex 8, and the escape character to hex 3.

The following fully describes all of the TTYSET parameters available to the user. Their initial values are defined, as well as any special characteristics they may possess.

BS=hh BackSpace character

This sets the 'backspace' charcter to the character having the ASCII hex value of hh. This character is initially a 'control H' (hex 08), but may be defined to any ASCII character. The action of the backspace character is to delete the last character typed from the terminal. If two backspace characters are typed, the last two characters will be deleted, etc. Setting BS=0 will disable the backspace feature.

BE=hh Backspace Echo character

This defines the character to be sent to the terminal after a 'backspace' character is received. The character printed will have the ASCII hex value of hh. This character is initially set to a null but can be set to any ASCII character.

The BE command also has a very special use that will be of interest to some terminal owners, such as SWTPC CT-64.

If a hex 08 is specified as the echo character, FLEX will output a space (20) then another 08. This feature is very uesful for terminals which decode a hex 08 as a cursor left but which do not erase characters as the cursor is moved.

Example: Say that you mis-typed the word cat as shown below:
+++CAY

typing in one CTRL-H (hex 08) would position the cursor on top of the Y and delete the Y from the DOS input buffer. FLEX would then send out a space ($20) to erase the Y and another 08 (cursor left) to re-position the cursor.

DL=hh DeLete character

This sets the 'delete current line' character to the hex value hh. This character is initially a 'control X' (hex 18). The action of the delete character is to 'erase' the current input line before it is accepted into the computer for execution. Setting DL=0 will disable the line delete feature.

EL=hh End of Line character

This character is the one used by FLEX to separate multiple commands on one input line. It is initially set to a colon (':'), a hex value of 3A. Setting this character to 0 will disable the multiple command per line capability of FLEX. The parameter 'EL=hh' will set the end of line character to the character having the ASCII hex value of hh. This character must be set to a printable character (control characters not allowed).

DP=dd DePth count

This parameter specifies that a page consists of dd (decimal) physical lines of output. A page may be considered to be the number of lines between the fold if using fan folded paper on a hard copy terminal, or a page may be defined to be the number of lines which can be displayed at any one time on a CRT type terminal. Setting DP=0 will disable the paging (this is the initial value). See EJ and PS below for more details of depth.

WD=dd WiDth

The WD parameter specifies the (decimal) number of characters to be displayed on a physical line at the terminal (the number of columns). Lines of text longer than the value of width will be 'folded' at every multiple of WD characters. For example, if WD is 50 and a line of 125 characters is to be displayed, the first 50 characters are displayed on a physical line at the terminal, the next 50 characters are displayed on the next physical line, and the last 25 characters are displayed on the third physical line. If WD is set to 0, the width feature will be disabled, and any number of characters will be permitted on a physical line.

NL=dd NuLl count

This parameter sets the (decimal) number of non-printing (Null) 'pad' characters to be sent to the terminal at the end of each line. These pad characters are used so the terminal carriage has enough time to return to the left margin before the next printable characters are sent. The initial value is 4. Users using CRT type terminals may want to set NL=0 since no pad characters are usually required on this type of terminal.

TB=hh TaB character

The tab character is not used by FLEX but some of the utilities may require one (such as the Text Editing System). This parameter will set the tab character to the character having the ASCII hex value hh. This character should be a printable character.

EJ=dd EJect count

This parameter is used to specify the (decimal) number of 'eject lines' to be sent to the terminal at the bottom of each page. If Pause is 'on', the 'eject sequence' is sent to the terminal after the pause is terminated. If the value dd is zero (which it is by default), no 'eject lines' are issued. An eject line is simply a blank line (line feed) sent to the terminal. This feature is especially useful for terminals with fan fold paper to skip over the fold (see Depth). It may also be useful for certain CRT terminals to be able to erase the previous screen contents at the end of each page.

PS=Y or PS=N PauSe control

This parameter enables (PS=Y) or disables (PS=N) the end-of-page pause feature. If Pause is on and depth is set to some nonzero value, the output display is automatically suspended at the end of each page. The output may be restarted by typing the 'escape' character (see ES description). If pause is disabled, there will be no end-of-page pausing. This feature is useful for those using high-speed CRT terminals

to suspend output long enough to read the page of text.

ES=hh EScape character

The character whose ASCII hex value is hh is defined to be the 'escape character'. Its initial value is $1B, the ASCII ESC character. The escape character is used to stop output from being displayed, and once it is stopped, restart it again. It is also used to restart output after Pause has stopped it. As an example, suppose you are LISTing a long text file on the terminal and you wish to temporarily halt the output. Typing the 'escape character' will do this (this feature is not supported on computers using a Control Port for terminal communications). At this time (output halted), typing another 'escape character' will resume output, while typing a RETURN key will cause control to return to FLEX and the three plus sign prompt will be output to the terminal. It should be noted that line output stopping always happens at the end of a line.

VERIFY

The VERIFY command is used to set the File Management System's write verify mode. If VERIFY is on, every sector which is written to the disk is read back from the disk for verification (to make sure there are no errors in any sectors). With VERIFY off, no verification is performed.

DESCRIPTION

The general syntax of the VERIFY command is:

```
VERIFY[,ON]
   or
VERIFY[,OFF]
```

where ON or OFF sets the VERIFY mode accordingly. If VERIFY is typed without any parameters, the current status of VERIFY will be displayed on the terminal. Example:

```
+++VERIFY,ON
+++VERIFY
```

The first example sets the VERIFY mode to ON. The second line would display the current status (ON or OFF) of the VERIFY mode. VERIFY causes slower write times, but it is recommended that it be left on for your protection.

UNDELETE

+++UNDELETE[,<drive>]

UNDELETE attempts to restore deleted files from the free chain back into the directory with a user specified name.
The program starts with a search through the free chain and comes up with an interactive part to enable you some research on the deleted files. The most recently deleted file will be the first to be presented.
The commands are:

D Dump the current file in hex and ascii.
N Proceed to the next (older) file.
P Go back to the previous (younger) file.
R Recover the current file, and specify the filename.
S Stop, return to FLEX.

VALIDATE

+++VALIDATE[,<drive>]

This program performs a very clever check on the diskette to make sure everything is ok. Other disk diagnostics test if the sectors can be read. (Hardware check). VALIDATE tests if the structural information in the sector links and in the System Information Record are valid. (Software check).

VER

+++VER,<binary file>

This is the SWTPc version of VERSION by TSC.
VERSION will display a 1 byte version number, VER will give you a 5 byte string and date from the directory.
See VERSION

WALK0

***WALK0[,<starting address>,<ending address>]

The WALK0 memory test performs a "walking zero" test on a block of memory. At any time during the running of the test, only one bit in the entire block of memory being tested is a zero.
If no arguments were given, memory from 0 to FLEX "memory end" is tested.
See also WALK1

WALK1

+++WALK1[,<starting address>,<ending address>]

The WALK1 memory test performs a "walking one" test on a block of memory. At any time during the running of the test, only one bit in the entire block of memory being tested is a zero.
If no arguments were given, memory from 0 to FLEX "memory end" is tested.
See also WALK0

VERSION

The VERSION utility is used to display the version number of a utility command. If problems or updates ever occur in any of the utilities, they may be replaced with updated versions. The VERSION command will allow you to determine which version of a particular utility you have.

DESCRIPTION

The general syntax of the VERSION command is:

```
VERSION,<file spec>
```

where <file spec> is the name of the utility you wish to check. The default extension is CMD and the drive defaults to the working drive. As an example:

```
+++VERSION,0.CAT
```

would display the version number of the CAT command (from drive 0) on the terminal.

XOUT

XOUT is a special form of the delete command which deletes all files having the extension .OUT.

DESCRIPTION The general syntax of XOUT is:

```
XOUT[,<drive spec>]
```

where <drive spec> is the desired drive number. If no drive is specified all, .OUT files on the working drive will be deleted and if auto drive searching is enabled, all .OUT files on drives 1 and 2 will be deleted. XOUT will not delete any files which are delete protected or which are currently in the print queue.

Example:

```
+++XOUT
+++XOUT 1
```

XBASIC

+++XBASIC [<file spec>]

Where <file spec> defaults to the .BAC extension and the work drive.

Extended basic, is gelijk aan RS extended basic met de volgende uitzonderingen :

CHAIN "name" [ln]

laad en execute een subprogramma van disk, start optioneel op regelnummer ln

COMPILE "name"

schrijft progr. in een kortere form naar disk met extensie .BAC, de file kan niet meer gelist worden, alleen RUN "name" vanuit XBASIC of XBASIC name vanuit FLEX

DIGITS xx[,yy]

zet het aantal digits op xx voor de punt en yy na de punt van getallen, max 17

DPEEK (n)
DPOKE n,m

is een dubbele peek resp poke

EXIT

gaat uit XBASIC en uit FLEX !!!!!

EXEC "srting"

is hetzelfde als het EXEC commando van FLEX

FLEX

terug naar FLEX

FRE (0)

geeft vrije geheugen is gelijk aan MEM

GET #n[,RECORD nn]

leest uit file nr. n het volgende record, of optionaal record nn

HEX ("xxxx")

geeft de decimale waarde van hex getal xxxx

INCH$ [(n)]

is gelijk aan INKEY$, behalve dat er optionaal een device nr. gegeven mag worden als (n)

INPUT LINE ["text";] var$

is gelijk aan LINE INPUT

LSET O$=N$

overschrijft O$ met N$ waabij de lengte van O$ wordt aangehouden, eventueel rechts aangevult met spaties

ON ERROR GOTO ln

gaat bij een error naar regelnr ln, als ln=0 wordt de error handler gereset

OPEN [OLD/NEW] "name" AS n

opent een file op disk, n is filenummer, als OLD wordt meegegeven dan wordt een bestaande file geopent, als NEW wordt meegegeven dan wordt een nieuwe file gecreeerd, als ze reeds bestaat wordt ze gewist file nrs 1 tot 12

PI

geeft de waarde pi, 3,141592654

PUT #n[,RECORD nn]

schrijft naar disk als GET

PTR (var$)

is gelijk aan VARPTR

RESUME [NEXT]

gaat na een error welke is getrapt door een ON ERROR GOTO terug naar de plaats van de error, of als NEXT gegeven voorbij de error

RND (n)

RSET O$=N$

als LSET, doch nu aangevult links met spaties

als n<0 dan word een nieuwe serie seed gegenereerd
n=0 genereerd een nieuw random nummer

n>0 geeft vorig random nummer

SCALE x

set aantal digits na dec. punt, max 6

SPC (n)

in een PRINT commando, print n spaties

USR (n)

machine taal aanroep, de waarde n wordt op geheugen lokatie memend-4 geplaatst, het beginadres van de roetine moet al op geheugen lokatie memend-2 zijn gepoked, memend staat in lokaties $CC2B en CC2C

+ commado

doet een FLEX commando, b.v. +RENUMBER

PRINT #n, / INPUT #n, zijn ook aanwezig en gelijk

variabelen met % erachter zijn integer variabelen, b.v. A%

controle toetsen: C=break, H=←, X=shift ←

RENUMBER

+++RENUMBER,[<start line >],[<increment>]

This is called from BASIC and will renumber the program in memory the default are 10,10.

16. INDEX TO STATEMENTS AND COMMANDS

17. ERROR SUMMARY

Any time a program that is being executed encounters an error, of any kind, execution will be halted immediately and an ERROR MESSAGE will be printed (except when an ON ERROR is in effect). The message contains an ERROR NUMBER which can be looked up in the following table and also the line number in which the error occurred. The table provides a brief explanation of what type of error the number represents. An example of an error message that you could receive is:

ERROR 50 AT LINE 100

Looking in the ERROR TABLE, we see that error number 50 represents an "unrecognizible statement". The message tells us that it occurred at line 100 so we could do a "LIST 100" to display this line and we will more than likely find a typing error in it. The error should be corrected then the program can be run again.

All errors are assigned numbers below 100 except the arithmetic errors which range from 101 through 109 and error number 255. Error 255 informs you that an illegal token has been encountered. This error should never be encountered during normal program debugging. Its occurrence indicates the presence of a bad memory location or other serious problems.

The errors are divided into two tables. Table one contains all of the I/O related errors and are numbered 1 through 49. It is this set of errors which may be acted upon by using the ON ERROR statement. It should be noted that all errors below error number 30 are FLEX errors and their numbers are identical to the FLEX error numbers. Errors 50 through 99 are related to syntax or computational type errors.

NUMBER	MEANING
1	ILLEGAL FMS FUNCTION CODE
2	THE REQUESTED FILE IS IN USE
3	THE FILE ALREADY EXISTS
4	THE FILE COULD NOT BE FOUND
7	ALL DISK SPACE HAS BEEN USED
8	END OF FILE ERROR
9	DISK FILE READ ERROR
10	DISK FILE WRITE ERROR
11	THE FILE OR DISK IS WRITE PROTECTED
12	THE FILE IS PROTECTED
15	ILLEGAL DRIVE NUMBER SPECIFIED
16	DRIVES NOT READY
21	ILLEGAL FILE SPECIFICATION
22	FILE CLOSE ERROR
23	SECTOR MAP OVERFLOW
24	NON-EXISTENT RECORD NUMBER SPECIFIED
25	RECORD NUMBER MATCH ERROR - FILE DAMAGED
26	FLEX COMMAND ERROR
30	DATA TYPE MISMATCH
31	OUT OF DATA IN "READ"
32	BAD ARGUMENT IN "ON" STATEMENT
34	PROGRAMMABLE BREAK (CONTROL-C) TRAP
37	FLEX "ESCAPE RETURN" SEQUENCE TRAP
40	BAD FILE NUMBER USED
41	FILE ALREADY OPEN
42	MUST OPEN FILE AS "NEW" OR "OLD"
43	FILE HAS NOT BEEN OPENED
44	FILE STATUS ERROR
45	FIELD SIZE ERROR (>252 OR <0)
46	CAN'T EXTEND A SEQUENTIAL FILE
47	RECORD 0 NOT ALLOWED
48	MUST USE RANDOM TYPE FILE
50	UNRECOGNIZABLE STATEMENT
51	ILLEGAL CHARACTER IN LINE
52	SYNTAX ERROR
53	ILLEGAL LINE TERMINATION
54	LINE NUMBER 0 NOT ALLOWED
55	UNBALANCED PARENTHESES
56	ILLEGAL FUNCTION REFERENCE
57	MISSING QUOTE IN STRING CONSTANT
58	MISSING "THEN" IN AN "IF" STATEMENT

NUMBER	MEANING
60	LINE NOT FOUND
61	RETURN WITHOUT "GOSUB"
62	"FOR-NEXT" NEST ERROR
63	CAN'T CONTINUE
64	SOURCE NOT PRESENT
65	BAD FILE - WON'T LOAD
66	"RESUME" NOT IN ERROR ROUTINE
67	CAN'T CHANGE SCALE FACTOR
70	DATA TYPE MISMATCH IN "PRINT USING"
71	ILLEGAL FORMAT IN "PRINT USING"
72	MIXED MODE IN AN EXPRESSION
73	ILLEGAL EXPRESSION
74	ARGUMENT <0 OR >255
75	ARGUMENT >32,767
76	ILLEGAL VARIABLE TYPE
77	ARRAY REFERENCE OUT OF RANGE
78	UNDIMENSIONED ARRAY REFERENCE
79	BAD ARGUMENT IN "SWAP" STATEMENT
80	MEMORY OVERFLOW
81	ARRAY OVERFLOW
83	STRING TOO LONG
90	UNDEFINED USER FUNCTION
91	UNDEFINED USER CALL
94	BAD STRING LENGTH SPECIFIED
100	EXPRESSION TOO COMPLEX
101	OVERFLOW OR UNDERFLOW IN FLOATING POINT OP.
102	ARGUMENT TOO LARGE
103	DIVISION BY ZERO
104	NUMBER TOO LARGE TO CONVERT TO INTEGER
105	NEGATIVE OR ZERO ARGUMENT FOR "LOG"
106	CONVERSION ERROR IN INTEGER "INPUT"
107	IMAGINARY SQUARE ROOT
108	CONVERSION ERROR (NUMBER TOO LARGE)
109	OVERFLOW/UNDERFLOW IN INTEGER OPERATION
255	ILLEGAL TOKEN ENCOUNTERED

X5124BW
X5124WB
X6424BW
X6424WB
X6432BW
X3216BW

High resolution output screens

Xnnnnzz is used to invoke the different Hi-res screens

DESCRIPTION

The general syntax of the Xnnnnzz command is:

X5124BW

The variations are:

X**5124**BW = **51** columns by **24** lines, black characters on white.

X**5124**WB = **51** columns by **24** lines, white characters on black.

X**6424**BW = **64** columns by **24** lines, black characters on white.

X**6424**WB = **64** columns by **24** lines, white characters on black.

X**6432**BW = **64** columns by **32** lines, black characters on white.

X**3216**BW = **32** columns by **16** lines, black characters on white.

The default is X5124BW. You can append any of these to FLEX and have it come up in the type that you prefer. See the SETUP command under the 'F' option for how to append things to FLEX.

All of these screens use the hi-res screen. Some of them (6424 and 6432) may only be readable on a monitor. Your individual TV will make the difference. A B&W TV is better than a color set. But if you turn the color down on your color set the characters are easier to read.

NOTE: Hi-Res screen mode cannot be changed by using an Xcommand unless you are in a Color Computer Screen mode (not EXT). Also, MEMEND must not be changed. The error message is "CANNOT BE RUN IN THIS MODE."

Y

The Y (Yes) utility allows you to determine how a yes/no prompt from another utility will be answered.

DESCRIPTION

The general syntax of the Y command is:

Y,<COMMAND>

where the command is any applicable utility command file.

An example of this is:

+++Y,DELETE,BADFILE.TXT

This will answer "Yes" to the "DELETE BADFILE.TXT?" and then yes again to "ARE YOU SURE?"

As you can probably see, you should exercise care in using this utility.

ZAP

The ZAP command is a file delete utility. Either all files or only files matching a specified match list are deleted without any prompting. This command is very convenient for quickly removing a lot of no longer needed files from a disk.

DESCRIPTION

The general syntax of the ZAP command is:

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

where drive list and match list are the same as described in the CAT command. Upon execution of ZAP, the name of each file deleted will be printed at the terminal in the form:

```
DELETING "FILE"
```

Be aware that there is no chance for "second thoughts". Once ZAP is invoked, the files will be deleted without any further intervention by the user. An example follows:

```
+++ZAP,1,.BAK
```

This command would cause all of the files on drive 1 with a .BAK extension to be deleted. It is wise, before invoking ZAP, to check which files will be deleted by doing a CAT, DIR, or FILES with the same match list that will be used with ZAP.