

# **6809 FLEX™** **Operating** **System**



technical systems  
consultants, inc.

# **The FLEX<sup>™</sup> Disk Operating System**

**Technical Systems Consultants, Inc.**

#### COPYRIGHT INFORMATION

This entire manual is provided for the personal use and enjoyment of the purchaser. Its contents are copyrighted by Technical Systems Consultants, Inc., and reproduction, in whole or in part, by any means is prohibited. Use of this program, or any part thereof, for any purpose other than single end use by the purchaser is prohibited.

#### DISCLAIMER

The supplied software is intended for use only as described in this manual. Use of undocumented features or parameters may cause unpredictable results for which Technical Systems Consultants, Inc. cannot assume responsibility. Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Technical Systems Consultants, Inc. will not assume responsibility for any damages incurred or generated by such material. Technical Systems Consultants, Inc. reserves the right to make changes in such material at any time without notice.

# **FLEX User's Manual**

**COPYRIGHT © 1979 by  
Technical Systems Consultants, Inc.  
P.O. Box 2570  
West Lafayette, Indiana 47906  
All Rights Reserved**

™ FLEX is a trademark of Technical Systems Consultants, Inc.

#### **COPYRIGHT INFORMATION**

This entire manual is provided for the personal use and enjoyment of the purchaser. Its contents are copyrighted by Technical Systems Consultants, Inc., and reproduction, in whole or in part, by any means is prohibited. Use of this program, or any part thereof, for any purpose other than single end use by the purchaser is prohibited.

#### **DISCLAIMER**

The supplied software is intended for use only as described in this manual. Use of undocumented features or parameters may cause unpredictable results for which Technical Systems Consultants, Inc. cannot assume responsibility. Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Technical Systems Consultants, Inc. will not assume responsibility for any damages incurred or generated by such material. Technical Systems Consultants, Inc. reserves the right to make changes in such material at any time without notice.

## PREFACE

The purpose of this User's Guide is to provide the user of the FLEX Operating System with the information required to make effective use of the available system commands and utilities. This manual applies to FLEX 9.0 for full size and mini floppy disks. The user should keep this manual close at hand while becoming familiar with the system. It is organized to make it convenient as a quick reference guide, as well as a thorough reference manual.



# FLEX USER'S MANUAL

## I. INTRODUCTION

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

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

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

---

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



## II. SYSTEM REQUIREMENTS

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

## III. GETTING THE SYSTEM STARTED

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

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

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

```
+++
```

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

#### IV. DISK FILES AND THEIR NAMES

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

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

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

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

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

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

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

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

Some examples of file specifications with drive numbers follow:

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

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

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

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

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

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

## V. ENTERING COMMANDS

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

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

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

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

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

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

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

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

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

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

```
+++LIST TEXTFILE
```

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

## VI. COMMAND DESCRIPTIONS

There are two types of commands in FLEX, memory resident (those which actually are part of the operating system) and disk utility commands (those commands which reside on the disk and are part of the UCS). There are only two resident commands, GET and MON. They will be described here while the UCS is described in the following sections.

### GET

The GET command is used to load a binary file into memory. It is a special purpose command and is not often used. It has the following syntax:

```
GET[,<file name list>]
```

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

Again the '[' surround optional items. 'File spec' denotes a file name as described earlier. 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.

### MON

MON is used to exit FLEX and return to the hardware monitor system such as S-BUG. The syntax for this command is simply MON followed by the 'RETURN' key.

NOTE: to re-enter FLEX after using the MON command, you should enter the program at location CD03 hex.

## 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.

# GIMIX FLEX USER NOTES #1

The following programs in the TSC 6809 DISK DIAGNOSTICS package will not work with GIMIX FLEX.

PROGRAM NAME	NOTES
TEST.CMD	Will give errors for track zero on mini floppy double density disk. GIMIX FLEX versions 2.0 and 3.0.
VALIDATE.CMD	Will abort with 'INVALID SYSTEM INFO SECTOR' message. This only applies to mini floppies formatted for more than 40 tracks, mini's formatted double density and eight inch disks formatted for more than 77 tracks. GIMIX FLEX versions 1.0, 2.0 and 3.0.
COPYR.CMD	Same as 'TEST.CMD' above.
EXAMINE.CMD	Same as 'TEST.CMD' above.
FLAW.CMD	Same as 'TEST.CMD' above.
REBUILD.CMD	Same as 'TEST.CMD' above.

NOTE: This is because these programs were not written to handle double density mini's, mini's with greater than forty tracks and eight inch's with greater than seventy-seven tracks.

The following 6809 FLEX Utilities will not work with GIMIX versions of FLEX:

NAME	NOTES
DIR.CMD	This program can cause the system to hang and/or crash when used while print spooling. This is because of the way that the FLEX print spooler handles the stack. This program will function perfectly when the print spooler is not active. GIMIX FLEX versions 1.0, 2.0 and 3.0.

FLEX is a trademark of Technical Systems Consultants, West Lafayette, Indiana.

GIMIX is a registered trademark of GIMIX, Inc., Chicago, Illinois.



Page 1.2

Paragraph II (SYSTEM REQUIREMENTS)

GIMIX FLEX requires memory from location \$0000 through location \$3FFF. As well as memory from location \$C000 through location \$DFFF. The system assumes that the user has four disk drives connected to the controller. To set this differently see the 'SETUP' command.

Paragraph III (GETTING THE SYSTEM STARTED)

When using GMXBUG-09 with the Disk BOOT PROM or BOOT + VIDEO PROM installed, type 'U', followed by a carriage return in response to the GMXBUG-09 '>' prompt to boot the system.

Page 1.3

Paragraph IV (DISK FILES AND THEIR NAMES)

The actual number of sectors available to the user may vary according to the number of tracks formatted, number of sides formatted, size of the disk and whether the disk was formatted single or double stepped. Please see the 'FORMAT' and 'SETUP' commands for more information.

Page 2.1

Paragraph 7 (GENERAL SYSTEM FEATURES)

In addition to the features already mentioned GIMIX FLEX also gives the user the following additional features: Selection of stepping speed by drive, Selection of number of drives installed on the system, software write protection by drive, selection of single or double stepping (See the 'SETUP' and 'REPORT' commands), and auto selection of disk density.

PAGE 3.1

Paragraph I (DISK CAPACITY)

See addendum for Paragraph IV, Page 1.3

Paragraph V (ACCESSING DRIVES NOT CONTAINING A DISKETTE)

When attempting to access a mini floppy drive that has no disk in it one of the following things can occur: The system will hang and wait for the user to insert a disk into the drive or the system will come back with the 'DRIVES NOT READY' message.

Some disk drives generate a 'not ready' signal to the controller. For the drives that do not generate this signal the 'pseudo ready' should be enabled. Please see the hardware manual for a list of which drives generate this signal and how to enable it and how to enable the 'pseudo ready' feature.

NOTE: When using mini floppys with the 'Pseudo Ready' enabled, the system will not always come back with the not ready message.

#### Page 3.4

Paragraph VIII (FLEX OPERATING SYSTEM INPUT/OUTPUT SUBROUTINES)

GIMIX FLEX uses the GMXBUG-09 (SBUG-E compatible) INPUT/OUTPUT routines for total system compatibility.

#### Page 3.6

Paragraph IX (BOOTING THE FLEX DISK OPERATING SYSTEM)

If neither the BOOT PROM or the VIDEO + BOOT PROM is installed in the system the user must hand enter a bootstrap. A suitable boot program is included in the manual for the disk controller.

#### Page 3.7

Paragraph X (REQUIREMENTS FOR THE 'PRINT.SYS' DRIVER)

GIMIX FLEX, as supplied, includes two printer drivers and their source codes. List the file 'READ-ME' on the system disk for more information.

NOTE: GIMIX FLEX does not have a 'NEWDISK' command. This has been replaced with the 'FORMAT' command.

# TABLE OF CONTENTS

	Page
CHAPTER 1	1.1
I. Introduction	1.2
II. System Requirements	1.2
III. Getting the System Started	1.3
IV. Disk Files and Their Names	1.5
V. Entering Commands	1.7
VI. Command Descriptions	
CHAPTER 2	2.1
I. Utility Command Set	A.1
APPEND	A.2
ASN	B.1
BUILD	C.1
CAT	C.2
COPY	C.3
*CLEAN	C.4
*CHECKSUM	D.1
DATE	D.2
DELETE	E.1
EXEC	E.2
*EXTEND	F.1
*FORMAT	I.1
I	J.1
JUMP	L.1
LINK	L.2
LIST	N.1
*NAME	N.2
*N	O.1
O	P.1
P	P.2
PRINT	P.3
PROT	Q.1
QCHECK	R.1
RENAME	R.2
*REPORT	S.1
SAVE	S.2
STARTUP	S.3
*SETTIME	S.4
*SETUP	T.1
TTYSET	T.2
*TIME	U.1
*USEMPT	U.5
*UPDATE	V.1
VERIFY	V.2
VERSION	X.1
XOUT	Y.1
*YEAR	Y.2
*Y	

\* Programs supplied by GIMIX

CHAPTER 3		
I.	Disk Capacity	3.1
II.	Write Protect	3.1
III.	The 'RESET' Button	3.1
IV.	Notes on the P Command	3.1
V.	Accessing Drives Not Containing a Disk	3.1
VI.	System Error Numbers	3.2
VII.	System Memory Map	3.3
VIII.	Flex Input/Output Subroutines	3.4
IX.	Booting the Flex Disk Operating System	3.6
X.	Requirements for 'PRINT.SYS' driver	3.7
XI.	Hardware Configuration	3.10
XII.	Patching FLEX to use the Time-of-Day clock	3.11
CHAPTER 4		
I.	Command Summary	4.1

# **6809 FLEX<sup>TM</sup> Utilities**

**COPYRIGHT © 1979 by  
Technical Systems Consultants, Inc.  
111 Providence Road  
Chapel Hill, North Carolina 27514  
All Rights Reserved**

#### COPYRIGHT INFORMATION

This entire manual is provided for the personal use and enjoyment of the purchaser. Its contents are copyrighted by Technical Systems Consultants, Inc., and reproduction, in whole or in part, by any means is prohibited. Use of this program, or any part thereof, for any purpose other than single end use by the purchaser is prohibited.

#### DISCLAIMER

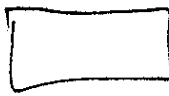
The supplied software is intended for use only as described in this manual. Use of undocumented features or parameters may cause unpredictable results for which Technical Systems Consultants, Inc. cannot assume responsibility. Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Technical Systems Consultants, Inc. will not assume responsibility for any damages incurred or generated by such material. Technical Systems Consultants, Inc. reserves the right to make changes in such material at any time without notice.

## INTRODUCTION

The utility programs documented herein are written to be position independent. Each program will run correctly, without modification, from any location in user memory. As a default, each program runs in the FLEX™ 9.0 Utility Command Space at location \$C100. The RUN utility may be used to load and execute the programs at other memory locations, if desired.

This manual is designed to be taken apart and the pages added to the "User's Manual" section of the "6809 FLEX™ Operating System" manual.

In addition to the command files, the source for these utilities is supplied on the diskette. Users may customize these programs for their own end use. However, the legal protection of the rights of Technical Systems Consultants, Inc. over the software extends, by law, to include such modified versions.

utilities  
marked  
with 

## 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.



## ASN

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
```

## 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.

## 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 separated 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

## 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.

## CHECKSUM

The CHECKSUM command performs a 32 bit checksum on an entire disk. The program reads every sector on the disk and totals them together. This can be used to verify disk copies, check disk validity, etc.

### DESCRIPTION

The general syntax of the CHECKSUM command is:

```
CHECKSUM[,dn]
```

Where 'dn' is an optional drive number. If no drive is specified CHECKSUM will use the work drive. If the work drive is set to 'ALL' an error message is printed. Some examples follow:

```
+++CHECKSUM  
+++CHECKSUM,2
```

The first example will generate a CHECKSUM of the disk in the current work drive, assuming the work drive is not set to 'ALL'. The second example will generate a CHECKSUM of the disk in drive 2. The output of CHECKSUM will look like:

```
CHECKSUM: 0002AB02
```

CHECKSUM can generate the following error messages:

#### ILLEGAL DRIVE NUMBER

Legal drive numbers are 0, 1, 2, or 3. A drive number must be specified if the work drive is set to ALL.

#### INVALID DISK FORMAT

The disk uses a non-standard format or the SYSTEM INFORMATION RECORD sector may be damaged.

## 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.

## 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.

## CLEAN

The CLEAN command has been provided to enable the user to use the head cleaning diskettes. All it does is step the head in and out to insure uniform cleaning.

### DESCRIPTION

The general syntax of the CLEAN command is:

#### CLEAN

CLEAN takes no command line parameters. It will prompt the user for the information that it needs.

To use the clean command merely type the following:

```
+++CLEAN
```

Clean will then prompt:

NUMBER OF TRACKS TO STEP?

Enter the maximum number of tracks for the drive to be cleaned as found in the manufacturer's literature. Though less than the maximum number of tracks may be specified, it is recommended that only the maximum number be used. This is to insure uniform head cleaning and uniform wear on the head cleaning diskette. Entering an illegal number or zero will cause a return to FLEX. The next prompt is:

NUMBER OF DRIVE TO BE CLEANED?

Enter the drive number for the drive to be cleaned. Entering an illegal number or an escape will cause a return to FLEX. The last prompt is:

PUT CLEANING DISK IN DRIVE AND HIT 'CR' TO CLEAN THE HEAD(S)?

At this point, follow the instructions that accompany the cleaning diskette. Insert the cleaning diskette in the specified drive and close the door. Then type a carriage return on the keyboard to start the cleaning process. Typing an escape will cause a return to FLEX. Typing any other character will cause the prompt to be re-printed. When finished CLEAN will print:

DONE.

And ring the terminal's bell.

NOTE: Failure to follow the manufacturer's instructions can cause damage to the disk drive and/or the cleaning diskette.

## 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 5,2,79 Sets the date register to May 2, 1979

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

Example:

```
+++DATE  
May 2, 1979
```

## 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".

```
1.00=ECHO, about to create a DUMSDISK in drive 1
2.00=FORMAT,1
3.00=COPY,0,1,COPY.CMD
4.00=COPY,0,1,DELETE.CMD
5.00=COPY,0,1,DIR.CMD
6.00=COPY,0,1,DUMSDISK.TXT
7.00=COPY,0,1,ECHO.CMD
8.00=COPY,0,1,EXEC.CMD
9.00=COPY,0,1,FORMAT.CMD
10.00=COPY,0,1,TEST.CMD
11.00=ECHO, finished creating DUMSDISK, now testing DUMSDISK
12.00=TEST,1
```



## 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.

## 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.

## 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.

## EXTEND

EXTEND enables the user to increase the amount of space allocated to the directory on a newly formatted disk. This prevents the directory from becoming fragmented when the number of directory entries exceeds the space allocated by the disk format program. Fragmenting the directory increases the amount of time required to access files on the disk.

### DESCRIPTION

The general syntax of the EXTEND command is:

```
EXTEND[,dn,sn]
```

Where `dn` is an optional drive number and `sn` is the number of additional sectors to be allocated to the directory. If no drive number is specified, EXTEND defaults to the work drive and adds 10 sectors. If the work drive is set to `ALL`, EXTEND prints an error message and returns to FLEX. The maximum number of additional sectors that can be allocated to the directory is 10. Each sector adds space for 10 entries to the directory allocation. Some examples follow:

```
+++EXTEND
+++EXTEND,2,5
```

The first example will EXTEND the directory of the disk in the work drive by 10 sectors (100 entries). The second example will EXTEND the directory of the disk in drive #2 by 5 sectors (50 entries).

The following table lists the number of sectors/directory entries normally allocated by FORMAT:

	SECTORS	5" ENTRIES	SECTORS	8" ENTRIES
SINGLE SIDED	6	60	11	110
DOUBLE SIDED	16	160	26	260

NOTE: EXTEND can only be used on a freshly formatted disk.

EXTEND can generate the following error message:

```
DISK CANNOT BE EXTENDED
```

Either there are already files on the disk or the first sector on track one was found bad when the disk was formatted.

## 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.

## 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'.



## 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.

## FORMAT

FORMAT is used to format a new diskette. Diskettes as purchased will not work with FLEX until certain formatting information has been put on them. The FORMAT utility writes this information on the diskette and then verifies that the information can be read back. If FORMAT finds sectors that it cannot read it removes them from the chain of free sectors and prints their location.

### DESCRIPTION

The general syntax of the FORMAT command is:

FORMAT[,<drive>]

Where <drive> is the number of the drive in which the disk to be formatted has been placed. If no drive number is specified the 'WORK' drive is used. If the 'WORK' drive is set to 'ALL' then the user is prompted for the drive number.

After FORMAT has determined the drive number it will ask:

SCRATCH DISK IN DRIVE #X ('Y' OR 'N')?

Where X is the drive number specified by the user or the 'WORK' drive. If the user types an 'N' the program will abort and return to FLEX. If a 'Y' is typed FORMAT continues with the following prompt:

DISK SIZE ('5' OR '8')?

The user then types in the size of the disk to be formatted. After this FORMAT prompts:

FORMAT SINGLE OR DOUBLE SIDED ('S' OR 'D')?

If the drive being used to format is a double sided drive and the user wants to format both sides of the disk type 'D', otherwise type 'S'. The next prompt is:

NUMBER OF TRACKS TO FORMAT?

FORMAT is asking literally how many tracks the user wishes to format. Standard sizes for 5.25" disks are: 35, 40, 70, 77, 80. Standard sizes for 8" disks are: 77, 154. The user can format less than the maximum number of sectors for special purposes. Please consult the disk drive manufacturers data sheet for the particular drive being used to find the maximum number of tracks that the drive is capable of accessing. Even though it is possible to attempt to format a disk for more tracks than it is capable of, it is not recommended as it might cause damage to the disk drive. The next prompt is:

FORMAT SINGLE OR DOUBLE STEPPING ('S' OR 'D')?

Some disk drives have double the normal number of tracks for that size drive. This is called a 'Double Tracking' disk drive. The Double Tracking drives have twice as many tracks per inch as regular disk drives. This makes them incompatible with regular drives unless the double stepping option is enabled (see the 'SETUP' command for more information). This option enables the user to create a disk that will be usable on a regular disk drive, but was formatted on a Double Tracking disk drive. This can be useful for program exchange, etc. The next prompt is:

1MHZ OR 2MHZ CPU SPEED ('1' OR '2')?

This tells FORMAT what speed the CPU is running at. This changes the 'interleave pattern' on the disk. The 'interleave pattern' is the order in which the sectors are put on the disk during formatting. The sectors are not placed in sequential order to enable the computer some processing time before having the next sector pass under the disk head. At faster speeds the computer has finished its processing and is waiting for the next sector to pass under the head. Changing the 'interleave pattern' to the 2MHz setting puts the sectors closer together on the disk so that the sector is there when the computer is ready for it. Disks formatted at the 2MHz setting can be used at 1MHz CPU speed, but they will take longer to read. The same goes for disks formatted at the 1MHz setting but being used at 2MHz CPU speed. When running at 1.5MHz select the 2MHz setting for mini-floppy disks and select the 1MHz setting for full size disks.

DISK NAME?

The user enters the name of the disk that is to appear in catalog listings. If the user just enters a carriage return a disk name of 'GIMIX .CHI' is put on the disk. The next prompt is:

VOLUME NUMBER?

The user enters the volume number that is to appear in catalog listings. If carriage return is entered then the disk number will be zero (0). If the user entered a carriage return for the name prompt, this prompt will be skipped and a volume number of '60609' will be put on the disk.

After entering the volume number FORMAT then prints all the data just entered and prompts:

IS THE ABOVE CORRECT ('Y' OR 'N')?

If the data typed in is correct then type 'Y' and FORMAT will go on. If an 'N' is typed then the prompts start over again. The final prompt is:

ABORT FORMAT ('Y' OR 'N')?

This is the users LAST chance to stop the formatting and save the disk in the specified drive. Typing an 'N' will start the format WITHOUT any further user interaction. Typing a 'Y' will abort the format and return to FLEX.

FORMAT will now print:

FORMATTING TRACK: XX

Where XX is the track currently being formatted. The track number will be updated as each track is formatted. After all tracks have been formatted 'FORMAT' will print:

VERIFYING TRACK: XX

Where XX is the track currently being verified. FORMAT reads every sector on the disk after formatting. If it finds a sector that it can not read it removes the bad sector from the chain of available sectors. A disk with a few bad sectors can still be used. Once a FORMAT has removed a sector it is unavailable to FLEX unless the disk is reformatted and does not error again. If a disk continually gives alot of errors or gives errors in different areas each time it is formatted the disk might be defective.

Upon successful completion FORMAT will print the following message and then return to FLEX:

FORMATTING COMPLETED  
TOTAL SECTORS: XXXX

Where XXXX is the number of sectors available to the user. This number will vary depending on the number of tracks formatted, the size of the disk, whether the disk was formatted single or double sided and whether the disk was formatted single or double stepped.

The following is an explanation of the possible error messages that can be generated by the FORMAT command:

NOT ENOUGH MEMORY INSTALLED IN SYSTEM

This means that according to the FLEX 'MEMEND' pointer there is not enough memory installed in the system to format a disk. The user must have at least sixteen (16) 'k' of memory starting at \$0000 in addition to the RAM occupied by FLEX.

FORMATTING ABORTED

This error message is printed to inform the user that FORMAT returned to FLEX prematurely and that formatting was unsuccessful.

#### TOO MANY TRACKS FOR DOUBLE STEPPING

This error message means that the user tried to format more tracks than any drive is capable of handling when double stepped. When formatting double stepped the number of tracks on the drive is HALVED. The user is then prompted for the number of tracks to format again.

#### ERROR WRITING BOOT SECTOR

This is a fatal error which causes the formatting to be aborted. This means that FORMAT could not put the necessary loading information on track 0, sector 1.

#### SECTOR WAS NOT WRITTEN TO ZEROS

This is a secondary error and is only printed after a bad sector message has been printed. It tells the user that the sector did not clear when initially written to disk. This can also indicate a memory failure. Before re-formatting the disk check the memory for proper operation. The memory range will be from \$100 to \$2FFF.

#### ERROR IN SECTOR LINKAGES

This is a secondary error and is only printed after a bad sector message has been printed. It tells the user that the pointers to the next sector were not written correctly. As mentioned above this can also indicate a memory error.

#### ERROR VERIFYING SECTOR

This is a secondary error and is only printed after a bad sector message has been printed. It tells the user that the specified sector cannot be read.

#### FATAL ERROR

This tells the user that FORMAT found an error in a vital area of the disk and that the disk is unusable.

#### BAD SECTOR AT: TT-SS

This is the header message for the three secondary error messages. TT is the track number of the error and SS is the sector number of the error. This message is printed if FORMAT finds a sector with an error in it during the verify operation.

#### NO GOOD SECTORS ON DISK

This is a fatal error and tells the user that FORMAT could not find a single usable sector on the disk. This usually means that the disk is defective. Try formatting the disk again before rejecting it.

#### DRIVE NOT READY

This tells the user that the drive to be formatted in either does not have a disk in it or that the drive door is open.

#### DISK IS WRITE PROTECTED

This is a fatal error that tells the user that the disk in the specified drive is write protected and cannot be formatted until it is un-write protected.

#### WRITE FAULT IN WRITING TRACK

This indicates a hardware failure in the disk drive itself. If this message is received, re-try the FORMAT and if it appears again the chances are that the disk drive is not functioning properly.

#### LOST DATA IN WRITING TRACK

This error should not normally occur. Since FORMAT inhibits the 'IRQ' and 'FIRQ' interrupts the only way to get this error message is if the system is getting 'NMI' interrupts or if running eight inch disks at 1MHz cpu speed and 'slow I/O' is enabled (see 'Hardware Configuration' for more information). Eliminate the source of the interrupts and try again. If this error persists or there are definately no interrupts being generated in your system then there might be a hardware failure. If using the GIMIX 6809 PLUS CPU BOARD with the 58167 Time-of-Day clock option installed, make sure that it is not enabled for 'NMI' interrupts. See the Hardware Manual for information on how to do this.

#### ERROR IN ACCESSING SYSTEM INFORMATION RECORD

This means that the format and verify went properly but after verifying the disk when FORMAT went to write the disk information on track 0, sector 3 it encountered an error. This is a fatal error.

NOTE: When formatting disk in double tracking drives for double

stepping use only fresh, i.e. never used, disks. Also when using double stepped disk in single tracking drives or single tracking disks in double stepped drives do not write to the disk. Writing to these disks can cause them to be unreadable on single tracking drives. This only applies to writing. The user can always read single tracked or double stepped disks.

## CREATING SYSTEM DISKETTES

A system disk is the one from which the operating system can be loaded. Normally the system disk will also contain the Utility Command Set (UCS). The following procedure should be used when preparing system disks.

1. Initialize the diskette using FORMAT as described on the preceding pages.
2. COPY all .CMD files desired to the new disk
3. Copy all .SYS files to the new disk. It should be noted that steps 2 and 3 can be done with one command; 'COPY,0,1,.CMD,.OV,.LOW,.SYS', assuming you are copying from drive 0 to drive 1 and all command files and their overlays are desired. (the .OV copies overlay files and .LOW copies the utility 'SAVE.LOW').
4. Last it is necessary to LINK the file FLEX.SYS to the system using the LINK command.

A very convenient way to get the above process performed without having to type all of the commands each time is to create a command file and use the EXEC command. Consult the EXEC documentation for details.

It is not necessary to make every disk a system diskette. It is also possible to create 'working' diskettes, disks which do not have operating system on them, for use with text files or BASIC files. Remember that a diskette can not be used for booting the system unless the operating system is contained on it and it has been linked. To create a working disk, simply run FORMAT on a diskette. It will now have all of the required information to enable FLEX to make use of it. This disk, however, does not contain the disk operating system and is not capable of booting the system.

## 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.

## 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 O.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.

## JUMP

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 tell the bootstrap loader where the FLEX operating system file resides on the disk. This is necessary each time a system disk is created using NEWDISK. The NEWDISK utility should be consulted for complete details on the use of LINK.

### DESCRIPTION

The general syntax of the LINK command is:

```
LINK,<file spec>
```

where <file spec> is usually FLEX. The default extension is SYS. Some examples of the use of LINK follow:

```
+++LINK,FLEX  
+++LINK,1.FLEX
```

The first line will LINK FLEX.SYS on the working drive, while the second example will LINK FLEX.SYS on drive 1. For more advanced details of the LINK utility, consult the "Advanced Programmers Guide".

## 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 interpreted 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.

## 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

The MEMEND command is used to interrogate or set the FLEX Memory End value. This value is used by many FLEX programs to determine the last memory location that they may use.

### DESCRIPTION

The general syntax of the MEMEND command is:

```
MEMEND?           or
MEMEND            or
MEMEND <value>
```

The first form, that ending with a question mark, will cause the current value of Memory End to be printed at the terminal. The second form will cause MEMEND to perform a non-destructive test of memory to determine the highest usable address. It is assumed that the computer's memory is organized in 4K blocks starting at location zero. The Memory End value is set to that value which was determined to be the last address of contiguous memory by the memory test; it is also printed at the terminal. The third form, with a parameter, will set the Memory End value to that specified in the parameter. The parameter must be a hex value, without a leading dollar sign. Some examples follow:

```
+++MEMEND?
+++MEMEND
+++MEMEND 7FFF
```

The first example will cause the current value of Memory End to be printed at the terminal. The second example will cause a test to be performed, and Memory End to be set to the value determined by the test as the end of contiguous memory. The third example will cause the value of Memory End to be set to hex 7FFF.

The MEMEND utility should be used to reset Memory End whenever another program has modified it. An example is the abnormal exit from an EXEC procedure. EXEC changes Memory End to protect itself and, if aborted, will not restore it to its previous value.

## NAME

The NAME utility enables the user to change the name, extension, volume number and date in the system information sector of a disk.

### DESCRIPTION

The general synatax of the NAME command is:

```
NAME[,dn]
```

Where 'dn' is an optional drive number. If no drive is specified NAME will use the work drive. If the work drive is set to 'ALL' an error message is printed. Some examples follow:

```
+++NAME  
+++NAME,2
```

The first example will change the information on the disk in the work drive, assuming that the work drive is not set to all. The second example will change the information on the disk in drive #2.

NAME prints the current disk name, extension, volume number and date and then prompts for the new name. The new name and extension should be entered, followed by a carriage return. Entering only a carriage return will retain the old name. NAME then prompts for the new volume number. The new volume number should be entered, followed by a carriage return. Entering only a carriage return will retain the original volume number. After the new name and volume number have been entered, NAME prompts:

```
CHANGE DATE ('Y' OR 'N')?
```

Entering 'Y' changes the date on the disk to the "current date",  
Entering 'N' retains the old date.

NAME can generate the following error message:

```
ILLEGAL DRIVE NUMBER
```

Legal drive numbers are 0, 1, 2, and 3. A drive number must be specified if the work drive is set to 'ALL'.

NOTE: If NAME is used in a command line with multiple commands, it must be the last command on the line.



## N

The N utility enables the user to automatically answer "N" (no) to "Y or N" prompts from other utilities. The N utility is especially useful when writing EXEC files.

### DESCRIPTION

The general synatax of the N command is:

N,<command string>

Where <command string> is a valid command line to be executed. If N is used in a line with multiple commands, using the end of line character, it only affects the command immediately following it. For example:

+++N,COPY,0,1

Will copy, from drive #0 to drive #1, only those files that do not already exist on drive #1, automatically answering "N" (no) to any "DELETE ORIGINAL?" and "ARE YOU SURE?" prompts that occur because of duplicate files on the two disks.

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

PDIR0

PDIR1

EXEC to print directory with header

EXEC, PDIR0

EXEC, PDIR1

file & utilities must exist on disk 0.

Page - 6240 of 1261

4-11-84

## TABLE 3

```

1.00=P;HECHO;1B;0E;2A;2A;2A;CA;2B;2B;44;47;53;4D;2D;46;49;52;45;43;C9;4F;5C
;E9;2D;2B;2F;2F;2F;2A;1B;0F;0A;1D
2.00=P;HECHO;1B;0E
3.00=P;DATE
4.00=P;HECHO;1B;0F;1A;0D
5.00=P;DIF

```

FOUO - EXERCISE ONLY

The reaction scheme shows the conversion of 2,4,6-trimethyl-3-pyridinecarboxylic acid (2) to 2,4,6-trimethyl-3-pyridinecarboxaldehyde (1) using DMSO and POCl<sub>3</sub>. The yield is 80%.

2,4,6-trimethyl-3-pyridinecarboxylic acid (2) + DMSO + POCl<sub>3</sub> → 2,4,6-trimethyl-3-pyridinecarboxaldehyde (1) + H<sub>2</sub>O

Yield: 80%

PAGE 1

```

1.00=P,HECHD,18,CE,2A,7A,2A,2A,2C,2B,44,49,23,4B,2C,44,49,22,4C,4B,24,4F,52,
+69,26,2C,2B,2B,2A,2A,18,4F,0A,0D
2.00=P,HECHD,18,DE
3.00=P,DATE
4.00=P,HECHD,18,0F,0A,0D
5.00=P,DLP,1

```

## P

The P command is very special and unlike any others 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 used with the LIST command will allow the printing of FLEX text files.

### 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 have affect on the command it immediately preceeds. 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. Consult the 'Advanced Programmer's Guide' for details concerning adaption of the P command to various printers.

The P command tries to load a file named PRINT.SYS from the same disk which P itself was retrieved. The PRINT.SYS file which is supplied with the system diskette contains the necessary routines to operate a SWTPC PR 40 printer connected through a parallel interface on PORT 7 of the computer. If you wish to use a different printer configuration, consult the 'Advanced Programmer's Guide' for details on writing your own printer driver routines to replace the PRINT.SYS file. The PR 40 drivers, however, are compatible with many other parallel interfaced printers presently on the market.

## PRINT

FLEX has the ability to output file stored data to a printer at the same time that it is performing other tasks. This feature is especially useful when it is necessary to print a long listing without tying up the computer. This method of printing is called PRINTER SPOOLING. In order for the printer spooling function to work, a SWTPC MP-T interrupt timer board must be installed in I/O position #4 on the computer's mother board.

### DESCRIPTION

The general syntax of the PRINT command is as follows:

```
PRINT,<file spec>[,<repeat #>]
```

where <file spec> is the name of the file to be printed. The default extension on <file spec> is .OUT. <Repeat #> is the number of additional copies of the file you wish to be printed.

For example, say that your disk had a very large number of files on it and a printer catalog listing was desired. A file containing the output information should first be created by using the O command such as:

```
+++O,CAT.OUT,CAT.CMD    or    +++O,CAT,CAT
      (see the description of the O command)
```

when printer output is desired the command

```
+++PRINT,CAT.OUT    or    +++PRINT,CAT
```

should be entered.

At this time the file CAT.OUT is stored in a buffer called a print queue (waiting list). If another PRINT command is issued before the first is finished, the second file will be in the next available location in the print queue.

After the file name to be printed has been stored in the print queue, control will return to the FLEX operating system. At this time you may perform any disk operation you want, such as deleting files, copying disks, etc. While you are using FLEX, PRINT will be outputting the desired file to the printer. PRINT will automatically wait for the printer to become ready (power up) even after the file has been entered into the print queue.

After printing the first file, the second file in the queue will be printed (if there is one), etc. The print queue may be examined or modified at any time by using the QCHECK utility.

NOTE: There are several things that the user should be aware of when using the printer spooling:

- 1) Any file that is in the print queue may not be deleted, renamed, or changed in any way until it has been printed or removed by the QCHECK print queue manager utility.
- 2) Disks which contain the files in the print queue should not be removed while the files are still in the queue.
- 3) The P command should not be used while files are waiting in the print queue.
- 4) Any paper or cassette tape load or any other operation which requires that the computer accept data at precise time intervals should not be executed during a printer spooling operation.

## PROT

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.



## 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!

## QCHECK

The QCHECK utility can be used to examine the contents of the print queue and to modify its 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 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 halt 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.

## 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 required 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.

## REPORT

The REPORT command enables the user check the current system configuration, as set by the 'SETUP' command. It reports the value currently set for CPU speed, the current last drive and the status of all drives on the system.

### DESCRIPTION

The general syntax of the REPORT command is:

REPORT

REPORT takes no parameters and prints the system status as defined by the defaults and the last SETUP command.

To use the REPORT command type the following:

+++REPORT

The output is self-explanatory. For more information see the SETUP command.

This command uses the FLEX output routines and may have its output re-directed by any of the FLEX output re-direction commands (i.e. 'P', 'O', etc.).

## 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.

## 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.



## 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.

## 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 parameters 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.

## SETTIME

The SETTIME command is provided so that the user may set the time on the Time-of-Day clock on the GIMIX 6809 PLUS CPU BOARD with the Time-of-Day clock option installed.

### DESCRIPTION

The general format of the SETTIME command is:

#### SETTIME

SETTIME takes no parameters and prompts the user for all pertinent information needed to set the clock.

To use the SETTIME command merely type the following:

```
+++SETTIME
```

The computer will then respond like this:

```
MINUTES (1 - 59)?
```

The user then types in the minutes to be set to. The program then proceeds to prompt the user for hours, day of the week, day of the month and month.

If the computer responds to any of the prompts with this message:

```
INVALID INPUT, PLEASE RE-TRY.
```

It means that you did not enter a valid input for that prompt.

This program uses the FLEX line buffer to enable the user to delete or backspace his entry before carriage return is typed. To correct an error after carriage return has been typed the user must re-execute the SETTIME command.

After the time has been entered and the following line is showing:

```
TYPE ANY CHARACTER TO START THE CLOCK?
```

The time on the clock will stay where it has been set to until a character is typed on the keyboard.

Since this command uses the FLEX line buffer it cannot be use in multiple statement lines unless it is the last statement on the line.

## SETUP

The SETUP command is provided to enable the user to define certain characteristics of his operating environment. Using this command the user may set the disk drive stepping speed (by drive), select whether a drive is single or double stepping, specify the last drive available on the system and inform FLEX of the CPU speed.

### DESCRIPTION

The general systax of the SETUP command is:

SETUP[,<parameter list>]

Where <parameter list> is a series of three and two letter parameters and drive number. If no parameters are given the the user is prompted with a menu.

The first character of each set of parameters is the drive number, 0 - 3. The next two letters and each subsequent two letters define a drive characteristic.

The parameters are defined as follows:

S0 = STEPPING SPEED OF 6ms FOR 8" (12ms FOR 5")  
S1 = STEPPING SPEED OF 6ms FOR 8" (12ms FOR 5")  
S2 = STEPPING SPEED OF 10ms FOR 8" (20ms FOR 5")  
S3 = STEPPING SPEED OF 20ms FOR 8" (40ms FOR 5")  
SS = SINGLE STEPPING  
DS = DOUBLE STEPPING  
LD = LAST DRIVE ON SYSTEM  
M1 = 1MHz CPU SPEED  
M2 = 1.5MHz CPU SPEED  
M3 = 2MHz CPU SPEED

The following describes these parameters in full detail.

#### S0, S1, S2 & S3

This tells the operating system what the stepping speed for the specified drive is. The default setting is 20ms (40ms for 5"). All four drives are set to this on boot.

When setting the drive stepping speed it is advisable to check the manufacturer specifications for each different drive on your system. This will prevent many errors from stepping a drive faster then it it is capable of.

## SS & DS

Some disk drives have double the normal number of tracks for that size drive. This is called a 'Double Tracking' disk drive. The Double Tracking drives have twice as many tracks per inch as regular disk drives. This makes them incompatible with regular drives. This command enables the user to step a Double Tracking drive twice for every normal step. This will make a Double Tracking drive appear to the system as a regular drive. The default setting is single stepping. All four drives are set to this on boot.

## LD

This tells the operating system how many drives are currently installed in the system. This protects the user from 'hanging' the system when inadvertently accessing a drive that is not actually on the system.

## M1, M2 & M3

This tells the system what the current CPU speed is. This is used for print spooling. The default is 1MHz. Please be sure to set this parameter if using a faster CPU speed to insure proper FLEX operation.

The following is an example of the use of this command:

```
+++SETUP,0S1,1S2SS,2S1DSLD  
+++SETUP
```

The first example tells the operating system the drive 0 is to step at 6ms (12 for 5"); drive 1 is to step at 10ms (20ms for 5") and be single stepping; drive 2 is to step at 6ms (12 for 5"), the drive is to be double stepped and is the last drive on the system. The second example will prompt the user for what he wants to change. The prompts are self-explanatory.

TEST

from FLEX diagnostics package -  
tests disk

TEST, 0  
TEST, 1

## 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' character 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 useful 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.

## TIME

The TIME command is provided so that the user may read the time on the Time-of-Day clock on the GIMIX 6809 PLUS CPU CARD with the Time-of-Day clock option installed.

### DESCRIPTION

The general syntax of the TIME command is:

TIME

TIME takes no parameters and prints the time as read from the clock.

To use the TIME command merely type the following:

+++TIME

The computer output will have the following format:

FRIDAY SEPTEMBER 05, 09:44:41 AM

If the computer responds:

ERROR READING TIME, CLOCK NOT SET

It means that the program detected an invalid value from the clock and the clock needs to be set. To set the time use the SETTIME command.

This command uses the FLEX output routines and therefore the output can be re-directed with any of the FLEX output re-direction command (i.e. 'P', 'O', etc.).

If there is no Time-of-Day clock installed in your system this program may cause the CPU to loop infinitely. If this happens the only way to exit the loop is to press the 'RESET' button on the front panel.

The USEMPT command has been provided to allow the GIMIX FLEX print spooler to be used with a SWTPC MP-09 CPU card. GIMIX FLEX normally uses the 6840 on the GIMIX 6809+ CPU card. This utility requires that the user have an SWTPC MP-T interrupt timer in I/O slot 4 (\$E040) of his mother board. Once this command has been invoked the only way to remove it is to re-boot. This program uses 39 bytes of user RAM at the top of available memory. It will not run if the user has less than 24K installed in the system (including FLEX RAM).

## DESCRIPTION

The general syntax of the USEMPT command is:

USEMPT

USEMPT takes no command line parameters.

To use the USEMPT command merely type the following:

+++USEMPT

The USEMPT command will respond with one of the following two messages:

MP-T IS NOW INSTALLED IN THE PRINT SPOOLER.

Which informs the user that the MP-T is now installed and ready for use. This utility does not check to see if the MP-T is functioning or if it is even in the system. It is up to the user to insure that the MP-T is placed in I/O slot 4 (\$E040) and enabled for IRQ type interrupts. If your system uses 4 bytes per I/O address then please contact GIMIX for a special version of this command.

The other possible message is:

NOT ENOUGH MEMORY INSTALLED IN SYSTEM

This informs the user that the USEMPT command could not find any usable RAM above address \$3FFF.

## UPDATE

The UPDATE utility enables the user to change the date in a file's directory entry to the "current date".

### DESCRIPTION

The general syntax of the UPDATE command is:

UPDATE,<filespec>

Where <filespec> is the name of the file for which the date is to be changed. If the file extension is not specified, UPDATE defaults to an extension of .TXT. The file's directory entry is changed to reflect the current date. UPDATE does not alter the contents of the file itself.

## 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.

## 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

## YEAR

The YEAR command is used to display or change the year in the internal FLEX date register. This command is used when FLEX is patched to load the current day and month from the Time-of-Day clock on the GIMIX 6809 CPU board (see the section on patching FLEX to use the Time-of-Day clock). The YEAR command should be included in the STARTUP file to set the year when the system is booted.

### DESCRIPTION

The general syntax of the YEAR command is:

```
YEAR,[YY]
```

Where YY is the last two digits of the current year. If no year is entered the current year in the FLEX system date area will be printed.

To use the YEAR command type the following:

```
+++YEAR
```

or

```
+++YEAR,81
```

The first example prints the year in the FLEX date register. The second example sets the year to 1981.

The error message :

```
INVALID YEAR IN INPUT LINE
```

Indicates that an illegal value was entered for the year [YY].

The Y utility enables the user to automatically answer "Y" (yes) to "Y or N" prompts from other utilities. The Y utility is especially useful when writing EXEC files.

#### DESCRIPTION

The general synatax of the Y command is:

Y,<command string>

Where <command string> is a valid command line to be executed. If Y is used in a line with multiple commands, using the end of line character, it only affects the command immediately following it. Some examples follow:

```
+++Y,COPY,0,1
+++Y,DELETE,TESTFILE.CMD
```

The first example will copy all files from drive #0 to drive #1, automatically answering "Y" (yes) to any "DELETE ORIGINAL?" and "ARE YOU SURE?" prompts that occur because of duplicate files on the two disks. The second example will delete the specified file, automatically answering "Y" (yes) to the "DELETE <file spec.>?" and "ARE YOU SURE?" prompts. Use caution when using the Y utility, especially when files are being deleted, since it bypasses the normal protection against unintentionally deleting the wrong file.



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.

## GENERAL SYSTEM INFORMATION

### I. DISK CAPACITY

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

### II. WRITE PROTECT

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

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

### III. THE 'RESET' BUTTON

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

### IV. NOTES ON THE P COMMAND

The P command tries to load a printer driver file named PRINT.SYS from the same disk which P itself was retrieved. For the requirements of this file and on writing your own custom PRINT.SYS file, see the section on such later in this manual or consult the 'Advanced Programmer's Guide'.

### V. ACCESSING DRIVES NOT CONTAINING A DISKETTE

If an attempt is made to access a minifloppy not containing a diskette, the system will hang up attempting to read until a disk is inserted and the door closed. Alternatively, you could reset the machine and begin execution at the warm start location \$CD03.

## VI. SYSTEM ERROR NUMBERS

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

DISK ERROR #xx

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

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

For more details concerning the meanings of these error messages, consult the 'Advanced Programmer's Guide'.

## VII. SYSTEM MEMORY MAP

The following is a brief list of the RAM space required by the FLEX Operating System. All address are in hex.

0000 - BFFF	User RAM *Note: Some of this space is used by NEWDISK, COPY and other utilities.
C000 - DFFF	Disk Operating System
C07F	System stack
C100 - C6FF	Utility command space
CD00	FLEX cold start entry address
CD03	FLEX warm start entry address

For a more detailed memory map, consult the 'Advanced Programmer's Guide'.

## VIII. FLEX OPERATING SYSTEM INPUT/OUTPUT SUBROUTINES

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

### GETCHR at \$CD15

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

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

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

### PUTCHR at \$CD18

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

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

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

The processor's registers are affected as follows:

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

### PSTRNG at \$CD1E

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

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

ACC. A Changed during the operation



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

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

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

#### STAT at \$CD4E

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

## IX. BOOTING THE FLEX DISK OPERATING SYSTEM

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

If you are using a Southwest Technical Products disk system and the S-BUG monitor, there are bootstraps stored in this ROM which you can use. They are executed by simply typing a 'D' for the full size floppy or a 'U' for the mini floppy.

Those users of other hardware or monitor ROM should use the boot supplied with the hardware if compatible with FLEX. A sample boot (for the SWTPc mini system) is given here for reference.

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

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

## X. REQUIREMENTS FOR THE 'PRINT.SYS' PRINTER DRIVER

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

- 1) The driver must be in a file called PRINT.SYS
- 2) Three separate routines must be supplied, a printer initialization routine (PINIT at \$CCCC), a check ready routine (PCHK at \$CCD8), and an output character routine (POUT at \$CCE4).
- 3) When the POUT routine is called by FLEX, the character to be output will be in the A accumulator. The output routine must not destroy the B, X, Y, or U registers. PINIT may destroy any registers. PCHK may NOT alter any registers.
- 4) The routines MUST start at the addresses specified, but may be continued anywhere in memory if there is not room where specified. If placed elsewhere in memory, be certain they do not conflict with any utilities or programs which will use them.
- 5) All three routines must end with a return from subroutine instruction (RTS).

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

E01C PIA EQU \$E01C PIA ADDRESS FOR PORT #7

\*  
\* PRINTER INITIALIZATION (MUST BE AT \$CCCC)  
\*

CCCC			ORG	\$CCCC	MUST RESIDE AT \$CCCC
CCC0 86	3A	PINIT	LDA	#\$3A	SELECT DATA DIRECTION REG.
CCC2 B7	E01D		STA	PIA+1	BY WRITING 0 IN DDR CONTROL
CCC5 86	FF		LDA	#\$FF	SELECT ALL OUTPUT LINES
CCC7 B7	E01C		STA	PIA	PUT IN DATA DIRECTION REG.
CCCA 86	3E		LDA	#\$3E	SET UP FOR TRANSITION CHECKS
CCCC B7	E01D		STA	PIA+1	AND ENABLE OUTPUT REGISTER
CCCF 39			RTS		

\* PRINTER READY ROUTINE

CCD0 7D	E01C	PREADY	TST	PIA	RESET PIA READY INDICATION
CCD3 73	CCE3		COM	PFLAG	SET THE PRINTER READY FLAG
CCD6 39			RTS		

```

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

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

END

```

## Sample Drivers for Serial Printer

The following listing is a sample set of drivers for a serial type printer using an ACIA as its interface. This set of drivers is not supplied on disk. In order to use these drivers, you must type in the source and assemble it. If you have a serial printer, you will probably want to replace the parallel PRINT.SYS file on the disk with one containing these drivers.

```

1          *
2          * PRINT.SYS DRIVERS FOR GENERAL SERIAL PRINTER
3          * CHANGE ACIA EQUATE IF NECESSARY
4          *
5
6          E01C ACIA EQU $E01C ACIA ADDRESS FOR PORT #7
7
8          *
9          * PRINTER INITIALIZATION (MUST BE AT $CCCO)
10         *
11         CCC0 ORG $CCCO MUST RESIDE AT $CCCO
12         CCC0 86 13 PINIT LDA #$13 RESET ACIA
13         CCC2 B7 E01C STA ACIA
14         CCC5 86 11 LDA #$11 SET 8 BITS & 2 STOP
15         CCC7 B7 E01C STA ACIA
16         CCCA 39 RTS RETURN
17
18         *
19         * CHECK FOR PRINTER READY (MUST BE AT $CCD8)
20         *
21         CCD8 ORG $CCD8 PRINT TEST AT $CCD8
22         CCD8 34 04 PCHK PSHS B SAVE B ACC.
23         CCDA F6 E01C LDB ACIA GET STATUS
24         CCDD 56 RORB GET TDR BIT INTO
25         CCDE 56 RORB SIGN POSITION
26         CCDF 56 RORB
27         CCE0 35 04 PULS B RESTORE B ACC.
28         CCE2 39 RTS RETURN
29
30         *
31         * PRINTER OUTPUT CHARACTER ROUTINE (MUST BE AT $CCE4)
32         *
33         CCE4 ORG $CCE4 MUST RESIDE AT $CCE4
34         CCE4 34 04 POUT PSHS B SAVE B ACC.
35         CCE6 F6 E01C LDB ACIA GET STATUS
36         CCE9 57 POUT2 ASRB GET TDR BIT
37         CCEA 57 ASRB INTO CARRY
38         CCEB 24 F9 BCC POUT2 LOOP IF NOT READY
39         CCED 35 04 PULS B RESTORE B ACC.
40         CCEF B7 E01D STA ACIA+1 WRITE OUT THE CHAR.
41         CCF2 39 RTS RETURN
42
43         END

```

\* PRINT.BYS for image writer through port 42  
 \* at address \$E00E, \$E00F,  
 \* program copied from FLEX 6809 manual

```

E00E  acia    equ    $E00E    acia address for port 42
*
* printer initialization routine
*
0000      org    $0000    must reside at $0000
0000 86    10      pinit    rda    ##10    reset acia
0002 87    E00E      sta    acia
0005 86    11      lda    ##11    set 8 bits and 2 stop bits
0007 87    E00E      sta    acia
000A 89              rts          return
*
* check for printer ready; must be at $00D8
*
00D8      org    $00D8
00D8 84    04      pchk    psbs    b        save b acc
00DA F6    E00E      ldb     acia
00DB 56              norb          set for bit into
00DE 56              norb          sign position
00DF 56              norb
00E0 85    04      puls    b        restore b acc
00E2 89              rts
*
* printer output character routine (must be at $00E4)
*
00E4      org    $00E4
00E4 84    04      pout    psbs    b        save b acc
00E6 F6    E00E      pout2   ldb     acia    set status
00E9 57              asrb          get idr bit
00EA 57              asrb          into carry
00EB 24    F9      bcc     pout2    loop if not ready
00ED 85    04      puls    b        restore b acc
00EF 87    E00F      sta     acia+1    write out the char
00F2 89              rts          return
end

```

0 ERROR(S) DETECTED

SYMBOL TABLE:

acia E00E pchk 00D8 pinit 0000 pout 00E4 pout2 00E6

1-Jan-87  
 used with imagewriter  
 through port 4 or swtch  
 at 9600 baud.

file: PRINTNLF.TXT

OPT PAG

PARALLEL PRINTER DRIVER FOR  
SUPPRESSING LINE FEED

10-8-83 TSC ASSEMBLER PAGE 1

```

* PRINT.SYS
*
* THIS IS A NON-RELOCATABLE PRINTER DRIVER
* THAT INTERFACES A PARALLEL PRINTER WITH
* THE FLEX 'P' COMMAND.
*
* TO USE THIS CODE RENAME THE FILE 'PRINT.SYS'
* TO 'PRINTSR1.SYS' USING THE FLEX 'RENAME'
* COMMAND. THEN ASSEMBLE THIS PROGRAM.
* TO USE A DIFFERENT PORT THE USER MUST PASS
* THE ADDRESS IN THE COMMAND LINE.
* FOR EXAMPLE:
*
*+++ASMB,PRINTPLL,PRINT.SYS,+SLYW,+$E042
*
* THIS WILL ASSEMBLE THE FILE 'PRINT.TXT',
* PUT THE BINARY IN THE FILE 'PRINT.SYS',
* SUPPRESS THE LISTING, SOURCE AND WARNING
* MESSAGES AND DELETE OLD BINARY. PLEASE SEE
* THE TSC ASSEMBLER MANUAL FOR MORE INFORMATION
*
* On initialization this program outputs a linefeed,
* following that linefeeds are ignored. The first
* linefeed on initialization is required to enable
* the printer. After that linefeeds, if printed would
* result in double spacing. This routine is translated
* from the 6800 version, not a modification of the 6809
* parallel printer program supplied by GIMIX.
*
* PORT EQUATE
*
E01C PIA EQU $E01C 3A PORT ADDRESS
*
* INITIALIZE THE PIA
*
CCCC ORG $CCCC
CCCC 86 FF PINIT LDA #$FF SET DIRECTION FOR OUTPUT
CCC2 B7 E01C STA PIA STORE IN DATA DIRECTION REGISTER
CCC5 86 3E LDA #$3E SET UP STROBE
CCC7 B7 E01D STA PIA+1 STORE IN COMMAND REGISTER
CCCA 86 0A LDA #$0A load linefeed
CCCC B7 E01C OUTST6 STA PIA load CHR to output
CCCF 86 36 LDA #$36
CCD1 B7 E01D STA PIA+1
CCD4 86 3E LDA #$3E
CCD6 20 04 BRP STBCON
* CHECK FOR PRINTER READY
*
CCD8 ORG $CCD8
CCD8 7D E01D PCHK TST PIA+1 READY FOR DATA?
CCD8 39 RTS
CCDC B7 E01D STBCON STA PIA+1

```

```

CCDF 39          EXIT      RTS
CCE0 86  20      PRSLNK    LDA    #$20      print blank
CCE2 20  E8      BRA      OUTSTB
                *
                * PRINTER OUTPUT ROUTINE
                *
CCE4          ORG      $CCE4
CCE4 80  F2      POUT      BSR      PCHK      PRINTER READY?
CCE6 2A  FC      BPL      POUT      NO: WAIT FOR READY
CCE8 7D  E01C    POUT2     TST      PIA      CLEAR READY BIT
CCES 81  0A      CMPA     #$0A
CCED 27  F1      BEQ      PRSLNK      IF linefeed print blank
CCEF 20  DB      BRA      OUTSTB

```

0 ERROR(S) DETECTED



## XI. SYSTEM CONFIGURATION FOR GIMIX FLEX 1.0

The system should be configured for GMXBUG-09 or SBUG-E as required.

### GIMIX 6809 CPU BOARD

1. Select the desired CPU speed. See the CPU board manual for more information. When running at 1.5 or 2MHz the slow I/O, as on the GIMIX 6809 MOTHER BOARD, must be enabled. Please note that only the 68B09 will run at 2MHz.
2. If print spooling is to be used configure the 6840 programmable timer by jumpering clock gate #1 to ground and connecting the timer to the 'IRQ' line.

### GIMIX 5/8 DISK CONTROLLER

1. Enable the 'Pseudo Ready' option. See the DISK CONTROLLER manual for more information. 3/3  
A B
2. Set up for sixteen bytes per I/O address.
3. Set up for head load with motor on.

### MEMORY ADDRESSING

1. Address 8k of memory for \$C000 to \$DFFF and the remainder starting at \$0000 and up to \$BFFF. GIMIX FLEX 1.0 requires 16k of memory starting at \$0000 and ending at \$3FFF in addition to the 8k starting at \$C000 and ending at \$DFFF.

PATCHING FLEX TO USE THE TIME-OF-DAY CLOCK  
ON THE GIMIX 6809 CPU BOARD

When Flex is initialized (booted) it normally prompts the user for the current date. GIMIX FLEX disks include a file called DATEPAT.BIN which, when appended to FLEX.SYS, configures FLEX to read the day and month from the Time-of-Day clock on the GIMIX 6809 CPU board. Since the current year is not available from the Time-of-Day clock, a separate utility called YEAR is provided to allow the year to be set from a STARTUP file. Once this patch is implemented and the YEAR command is included in the STARTUP file, the FLEX internal date register will be set to the current date automatically when the system is booted. The patched FLEX will not prompt the user for the date unless the date from the Time-of-Day clock is invalid (the clock is not properly set).

To patch FLEX, put a copy of the GIMIX FLEX system disk in drive zero (0) and execute the following command lines:

```
+++APPEND,0.FLEX.SYS,0.DATEPAT.BIN,0.FLEXDATE.SYS
```

```
+++LINK,0.FLEXDATE.SYS
```

This creates a new file called FLEXDATE.SYS which is the patched version of FLEX. When creating backup disks, "LINK" FLEXDATE.SYS to create a disk that reads the day and month from the Time-of-Day clock. "LINK" FLEX.SYS will create a disk that prompts for the date.

The YEAR command (see page Y.1.1) should be included in a STARTUP file, to set the current year, when the patched FLEX is used.

If the date read from the Time-of-Day clock is invalid the following message is printed:

CLOCK NOT SET

and FLEX will prompt for the current date. After the date is entered FLEX returns without executing the STARTUP file. If this occurs the SETTIME utility should be used to set the Time-of-Day clock, and the system re-booted to execute the STARTUP file.

THE FOLLOWING PROGRAMS ARE EXCLUSIVE TO GIMIX FLEX AND ARE  
ON YOUR DISK BUT NOT DOCUMENTED IN THE GIMIX FLEX MANUAL.

PROGRAM NAME	DESCRIPTION
ASMBPAT.TXT	SOURCE CODE FOR THE PATCH TO MAKE THE TSC ASSEMBLER PRINT THE TIME, FROM THE TIME-OF-DAY CLOCK OPTION ON THE GIMIX 6809 PLUS CPU BOARD, WITH THE PAGE HEADER. SHOULD BE APPENDED TO ASMS.CMD. LIST IT FOR MORE INSTRUCTIONS. NOTE: THIS PATCH IS FOR VERSION #2 ONLY OF THE TSC MNEMONIC ASSEMBLER.
ASMBPAT.BIN	OBJECT CODE FOR THE ABOVE PATCH.
DATEPAT.BIN	FLEX PATCH TO GET THE DATE OFF OF THE 58167 TIME OF DAY CLOCK ON THE GIMIX 6809 CPU CARD. SEE THE SECTION ON PATCHING FLEX TO USE THE TIME-OF-DAY CLOCK FOR INSTRUCTIONS ON ITS USE.
NEWBOOT9.TXT	SOURCE CODE FOR THE GIMIX POSITION INDEPENDANT DISK BOOT. THIS BOOT NEED NOT BE RE-ASSEMBLED TO BE USED AT ANOTHER ADDRESS. LIST IT FOR MORE INFORMATION.
NEWBOOT9.BIN	OBJECT CODE FOR THE ABOVE PROGRAM
OS9.TXT	SOURCE CODE FOR THE OS9/GMXPUG-09 SOFTWARE SWITCHING PROGRAM. PLEASE LIST IT FOR INSTRUCTIONS.
OS9.CMD	COMMAND FILE FOR THE ABOVE FILE. EXECUTABLE FROM FLEX.
SBUGPAT.TXT	SOURCE CODE FOR THE PATCH TO MAKE THE FLEX 'MON' COMMAND WORK WITH SBUG-E. SHOULD BE APPENDED TO FLEX.SYS. LIST IT FOR MORE INSTRUCTIONS.
SBUGPAT.BIN	OBJECT CODE FOR ABOVE PATCH.
USEMPT.TXT	THIS IS THE SOURCE CODE FOR THE 'USEMPT' COMMAND. IT IS INCLUDED TO ENABLE THE USER TO PUT THE MP-T AT ANOTHER ADDRESS OTHER THAN \$E040 (PORT 4 AT 16 BYTES PER I/O SLOT). LIST IT FOR INSTRUCTIONS ON HOW TO RE-ASSEMBLE IT. SEE THE 'USEMPT' COMMAND IN YOUR GIMIX FLEX MANUAL FOR MORE INFORMATION.
PRINT.SYS	SERIAL PRINTER DRIVER FOR PORT 0, SIDE A (BASE ADDRESS \$E000).
PRINTPLL.SYS	PARALLEL PRINTER DRIVER FOR PORT 4. RTOR R (BASE ADDRESS \$ED92).

TO USE THIS DRIVER SEE PRINTPLL.TXT  
BELOW.

PRINT.TXT

SOURCE CODE FOR SERIAL PRINTER DRIVER  
CAN BE RE-ASSEMBLED TO USE ANY PORT  
ADDRESS. LIST IT FOR MORE INFORMATION.

PRINTPLL.TXT

SOURCE CODE FOR PARALLEL PRINTER DRIVER  
CAN BE RE-ASSEMBLED FOR ANY PORT ADDRESS.  
LIST IT FOR MORE INFORMATION OR FOR  
INSTRUCTIONS ON HOW TO INSTALL THE  
PARALLEL PRINTER DRIVER.

TIME.TXT

SOURCE CODE FOR TIME COMMAND.

SETTIME.TXT

SOURCE CODE FOR SETTIME COMMAND.

SEND9511.TXT

SOURCE CODE FOR 9511A STAND ALONE DRIVERS.

SEND9512.TXT

SOURCE CODE FOR 9512 STAND ALONE DRIVERS.

BASPAT.TXT

SOURCE CODE FOR THE PATCHES TO TSC 6809  
BASIC TO WORK WITH GMXBUG-09. ONLY  
NECESSARY WHEN USING THE VIDEO BASED  
VERSION OF GMXBUG-09. THIS PATCH  
VECTORS THE CONTROL/C TEST THROUGH  
GIMIX FLEX, THEREFORE ONCE PATCHED  
THE BASIC WILL WORK WITH BOTH TERMINAL  
AND VIDEO BASED SYSTEMS.

BASPAT.BIN

OBJECT CODE TO ABOVE PROGRAM. CAN BE  
APPENDED TO 'BASIC.CMD'.

XBASPAT.TXT

SOURCE CODE FOR THE PATCHES TO TSC 6809  
EXTENDED BASIC TO WORK WITH GMXBUG-09.  
ONLY NECESSARY WHEN USING THE VIDEO  
BASED VERSION OF GMXBUG-09. THIS PATCH  
VECTORS THE CONTROL/C TEST THROUGH GIMIX  
FLEX, THEREFORE ONCE PATCHED THE BASIC  
WILL WORK WITH BOTH TERMINAL AND VIDEO  
BASED SYSTEMS.

XBASPAT.BIN

OBJECT CODE TO ABOVE PROGRAM. CAN BE  
APPENDED TO 'XBASIC.CMD'.

READ-ME.TXT

THIS FILE.

NOTE: THE FOLLOWING PROGRAMS ARE SUPPLIED BY GIMIX. PLEASE  
CONTACT GIMIX WITH ANY QUESTIONS ON THESE PROGRAMS.

TIME.CMD, TIME.TXT, SETTIME.CMD, SETTIME.TXT, FORMAT.CMD,  
SETUP.CMD, REPORT.CMD, PRINT.SYS, PRINT.TXT, PRINTPLL.SYS,  
PRINTPLL.TXT, SEND9512.TXT, SEND9511.TXT, BASPAT.TXT,  
BASPAT.BIN, XBASPAT.TXT, XBASPAT.BIN, SBUGPAT.TXT, SBUGPAT.BIN,  
ASMBPAT.TXT, ASMBPAT.BIN, NEWBOOT9.TXT, NEWBOOT9.BIN,  
CLEAN.CMD, OS9.TXT, OS9.CMD, USEMPT.CMD, USEMPT.TXT,  
DATEPAT.BIN, Y.CMD, N.CMD, CHECKSUM.CMD, NAME.CMD,  
EXTEND.CMD, UPDATE.CMD AND READ-ME.TXT.

```

* USEMPT
*
* THIS PROGRAM ENABLES USERS OF GIMIX DISK CONTROLLERS
* TO USE THE PRINT SPOOLING FEATURE OF GIMIX FLEX WITHOUT
* HAVING A GIMIX 6809 CPU CARD. THIS PROGRAM USES A LITTLE
*
* USER RAM (ABOUT 40 BYTES) AND MUST BE RUN EACH TIME THAT
* FLEX IS BOOTED. ONCE BOOTED IT WILL STAY INTACT UNTIL
* EITHER REBOOTED OR THE TSC 'MEMEND' COMMAND IS RUN.
*
* TO USE THIS PROGRAM MERELY TYPE:
*
*+++USEMPT
*
* THE PROGRAM WILL FIND THE END OF MEMORY AND IF THERE
* IS MORE THAN 16K STARTING AT ZERO ($0000 - $3FFF) IT
* WILL INSTALL THE MP-T DRIVERS AND DISABLE THE 6840 DRIVE
*
* NOTE: USE OF THIS PROGRAM NEGATES THE FUNCTION OF THE
* PROCESSOR SPEED SELECTION IN THE 'SETUP' COMMAND.
*
* TO ASSEMBLE THIS FOR ANOTHER ADDRESS TYPE IN THE FOLLOWING
*
*+++ASMP,USEMPT,+SLOC,+XXXXX
*
* WHERE XXXX IS THE ADDRESS OF THE PORT THAT THE MP-T
* IS CURRENTLY INSTALLED.
*
* NOTE: THIS WILL AUTOMATICALLY DELETE THE OLD USEMPT.CMD
* FILE.
*
* EQUATES
*
E01C PORT EQU $E01C PORT ADDRESS
C100 UCS EQU $C100 FLEX UTILITY COMMAND SPACE
CC2B MEMEND EQU $CC2B FLEX MEMORY END POINTER
D387 PROGLO EQU $D387 START OF I/O AREA FOR INIT AND SER
*
CD03 WARMS EQU $CD03 FLEX WARN START
CD1E FSTRNG EQU $CD1E FLEX PRINT STRING WITH CR/LF
*
* START OF PROGRAM
*
C100 ORG UCS
C100 20 01 USEMPT BRA BEGIN BRANCH AROUND VERSION NUMBER
C102 01 FCB 1 VERSION NUMBER 1
C103 BE CC2B BEGIN LDX MEMEND GET MEMEND POINTER
C106 8C 4000 CMFX #$4000 ENOUGH MEMORY?
C109 2C 3F BLO MEMIIS NO: PRINT ERROR
*
* ENOUGH MEMORY NOW MOVE
* INTO MEMEND AREA.
*

```

E

RS.

NG

VICE

C108 108E C10C	LDY	#ENOMOV	POINT TO END OF AREA TO MOVE
C10F A6 A2	USEMP1 LDA	0,-Y	GET BYTE AND DECREMENT

MF-T PATCH FOR USE WITH GIMIX FL 10-8-83 TSC ASSEMBLER PAGE 2

C111 A7 B2	STA	0,-X	STORE IN USER RAM
C113 108C C1A6	CMPLY	#STRIMV	AT BEGINNING?
C117 26 F6	BNE	USEMP1	NO: CONTINUE
C119 30 1F	LEAX	-1,X	GIVE A LITTLE HEAD ROOM
C11B BF CC2B	STX	MEMEND	PUT MEMEND POINTER BACK

\*  
\* SET UP JUMP VECTORS  
\*

C11E 30 01	LEAX	1,X	POINT TO ACTUAL START OF PROGRAM
C120 86 7E	LDA	#7E	JUMP CODE
C122 B7 D387	STA	#D387	STORE IN 6840 STUFF
C125 BF D388	STX	#D388	SET UP JUMP
C128 30 88 16	LEAX	22,X	POINT TO TIMON
C12B B7 D395	STA	#D395	STORE IN 6840 STUFF
C12E BF D396	STX	#D396	SET UP JUMP
C131 30 04	LEAX	4,X	POINT TO TIMOFF
C133 B7 D399	STA	#D399	STORE IN 6840 STUFF
C136 BF D39A	STX	#D39A	SET UP JUMP
C139 30 06	LEAX	6,X	POINT TO INTRPT
C13B B7 D39F	STA	#D39F	STORE IN 6840 STUFF
C13E BF D3A0	STX	#D3A0	SET UP JUMP

\*  
\* PRINT MESSAGE AND RETURN TO FLEX  
\*

C141 8E C14F	LDX	#MPTINS	POINT TO MESSAGE
C144 B0 CD1E	FLXOUT JSR	PSTRNG	PRINT IT
C147 7E CD03	JMP	WARMS	RETURN TO FLEX WARM START

\*  
\* NOT ENOUGH MEMORY INSTALLED IN SYSTEM ERROR  
\*

C14A 8E C17D	NEMJIS LDX	#NEMMSG	POINT TO MESSAGE
C14D 20 F5	BRA	FLXOUT	OUTPUT IT AND RETURN TO FLEX

\*  
\* MESSAGES  
\*

C14F 4D 50 2D 54	MPTINS FCC	/MF-T IS NOW INSTALLED IN THE PRINT SPOOLER.
------------------	------------	--

/, #D, #A	
C153 20 49 53 20	
C157 4E 4F 57 20	
C15B 49 4E 53 54	
C15F 41 4C 4C 4E	
C163 44 20 49 4E	
C167 20 54 48 4E	
C16B 20 50 52 49	
C16F 4E 54 20 53	
C173 50 4F 4F 4C	
C177 4E 52 2E 00	
C17B 0A	
C17C 04	

C17D 07 4E 4F 54	NEMMSG FCC	4	7, /NOT ENOUGH MEMORY INSTALLED IN SYSTEM/, #D
------------------	------------	---	--

#A	
C181 20 45 4E 4F	
C18C 55 47 48 20	
C18F 40 45 4C 4F	
C18D 52 59 20 49	
C191 4E 53 54 41	

C195 4C 4C 45 44  
 C199 20 49 4E 20  
 C19D 53 59 53 54  
 C1A1 45 4D 0D 0A  
 C1A5 04

FCB 4

```
*
*
* START OF CODE TO BE MOVED
*
*
* INITIALIZE TIMER
*
```

C1A6 8E	E01E	STRMOV	LDA	#PORT+2	POINT TO PIA
C1A9 86	FF		LDA	##FF	SET FOR OFFLINE
C1AB 87	84		STA	PORT	FILE TO PORT
C1AD 86	3C		LDA	#40	CONFIGURE PORT DIRECTION
C1AE 87	0		STA	PORT	DATA TO PIA
C1B1 86	8F		LDA	##8F	INITIAL RATE OF COUNT
C1B3 43	84		STA	PORT	COUNT OF TIMER
C1B4 86	84		LDA	##84	CLEAR ANY PENDING INTERRUPTS
C1B5 86	8E		LDA	##8E	ENABLE INTERRUPTS
C1B7 87	01		STA	PORT	GIVE TO PIA
C1BB 39			RTS		RETURN TO CALLING PROGRAM

```
*
*
* TIMER ON ROUTINE
*
```

C1BC 86	04		LDA	##4	SET FOR 10MS
C1BE 20	02		BRA	TOFF2	OUTPUT AND RETURN

```
*
*
* TIMER OFF ROUTINE
*
```

C1C0 86	8F		LDA	##8F	RESET AND HOLD TIMER
C1C2 B7	E01E	TOFF2	STA	PORT+2	GIVE TO PORT
C1C5 39			RTS		RETURN TO CALLING PROGRAM

```
*
*
* HANDLE INTERRUPTS
*
```

C1C6 B6	E01E		LDA	PORT+2	CLEAR INTERRUPTS
C1C9 7E	C700		JMP	#C700	RETURN TO SPOOLER HANDLER
	C1CC	ENDMOV	END	*	END OF AREA TO MOVE
			END	USEMPT	

0 ERROR(S) DETECTED

## COMMAND SUMMARY

APPEND,<file spec>[,<file list>],<file spec>  
Default extension: .TXT  
Description page: A.1

ASN[,W=<drive>][,S=<drive>]  
Description page: A.2

BUILD,<file spec>  
Default extension: .TXT  
Description pge: B.1

CAT[,<drive list>][,<match list>]  
Description page: C.1

COPY,<file spec>,<file spec>  
COPY,<file spec>,<drive>  
COPY,<drive>,<drive>[,<match list>]  
Description page: C.2

CLEAN[,<drive spec>]  
Description page: C.3

CHECKSUM[,<drive spec>]  
Description page: C.4

DATE[,<mm,dd,yy>]  
Description page: D.1

DELETE,<file spec>{,<file list>]  
Description page: D.2

EXEC,<file spec>  
Default extension: .TXT  
Description page: E.1

EXTEND[,<drive spec>,<parameter>]  
Description page: E.2

FORMAT[,<drive spec>]  
Description page: F.1

GET,<file spec>{,<file list>]  
Default extension: .BIN  
Description page: G.1

I,<file spec>,<command>  
Default extension: .TXT  
Description page: I.1

JUMP,<hex address>  
Description page: J.1

LINK,<file spec>  
Default extension: .SYS  
Description page: L.1



LIST,<file spec>[,<line range>][,N]  
     Default extension: .TXT  
     Description page: L.2

MON  
     Description page: 1.7

NAME[,<drive spec>]  
     Description page: N.1

N,<command string>  
     Description page: N.2

O,<file spec>,<command>  
     Default extension: .OUT  
     Description page: O.1

P,<command>  
     Description page: P.1

PRINT,<file spec>  
     Default extension: .OUT  
     Description page: P.2

PROT,<file spec>[,<options>]  
     Description page: P.3

QCHECK  
     Description page: Q.1

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

REPORT  
     Description page: R.2

SAVE,<file spec>,<begin adr>,<end adr>[,<transfer adr>]  
     Default extension: .BIN  
     Description page: S.1

STARTUP  
     Description page: S.2

SETTIME  
     Description page: S.3

SETUP[,<parameter list>]  
     Description page: S.4

TTYSET[,<parameter list>]  
     Description page: T.1

TIME  
     Description page: T.2

USEMPT  
     Description page: U.1

UPDATE,<file spec>  
Default extension: .TXT  
Description page: U.5

VERIFY[,<ON or OFF>]  
Description page: V.1

VERSION,<file spec>  
Default extension: .CMD  
Description page: V.2

XOUT[,<drive spec>]  
Description page: X.1

YEAR[,YY]  
Description page: Y.1

Y,<command string>  
Description page: Y.2