Chapter 5
MICROCASSETTE AND DISK FILES

This chapter describes procedures for creating and accessing files on microcassettes and floppy disks (including the RAM disk) with BASIC. The types of file covered include program files, random access files and sequential access files. In reading this chapter, keep in mind that file management is a process which involves a number of interrelated commands and statements, each of which must be prepared with consideration for the others. Also be sure to specify file descriptors in accordance with the rules described in the "2.12 Files" section of Chapter 2.

A summary of procedures for handling errors occurring during disk access is included at the end of this chapter, together with a review of general precautions to be observed in using microcassettes and floppy disks.

5.1 Program Files

This section reviews the commands and statements used to manipulate program files. In specifying these commands/statements, remember that the disk drive which is currently logged in is assumed unless otherwise specified in <file descriptor>. Thus if BASIC was entered from the command line with the logged in drive A:, files would be saved to the RAM disk. If you had entered BASIC from the menu with BASIC resident you may not know which is the currently logged in drive. Also remember that the CP/M operating system will automatically assume that the file name extension is explicitly specified.

SAVE <file descriptor> [| ,A |]
| ,P |

This command writes the program in memory to the disk or microcassette under the file name specified in <file descriptor>. If neither the A nor P options are specified the program is written to the disk or microcassette in compressed binary format. If the A (ASCII) option is specified, the file is written as a series of ASCII characters. If the P (PROTECT) option is specified, the file is saved in encoded binary format. A program saved using the P (PROTECT) option cannot be listed or edited when it is later reloaded; therefore it is recommended that you also save an unprotected copy of the program for future listing or editing.

LOAD <file descriptor> [,R]

This command loads the program specified in <file descriptor> into memory from the disk. If the R option is specified, the program will be automatically executed as soon as loading is completed. Executing this command without the R option closes all files which are currently open; however, files will not be closed if the R option is specified. This makes it possible to chain programs which access the same data files.

RUN <file descriptor>[,R]

If <file descriptor> is omitted, this command executes the program which is currently in memory. Specifying <file descriptor> causes the specified program to be loaded into memory from the disk or microcassette (deleting any program currently in memory) and to be immediately executed. As with the LOAD command, all open files are closed upon execution of this command unless the R option is specified.

MERGE <file descriptor>

The MERGE command loads the specified program into memory from the disk or microcassette and merges it with the program in memory. The program merged must have been stored in ASCII format. If any lines of the program loaded have the same line numbers as those of the program in memory, those program lines in memory are replaced with those of the program from the disk or microcassette. BASIC always returns to the command level after execution of a MERGE command.

KILL <file descriptor>

This command deletes the specified file from the disk. The function of this command is the same regardless of whether the file specified is a system file, a program file, or a random or sequential access data file; therefore, great care should be exercised in using it.

NAME <old filename> AS <new filename>

This command is used to change the name of a file. Specify the current file name in <old filename> and the new file name which is to be assigned to the file in <new filename>. This command can be used to rename any type of file.

5.2 Sequential Files

This section describes procedures for creating, accessing and updating sequential data files. Sequential files are easier to create than random files, but they are not as easy to update and take longer to access. As the name implies, the items included in a sequential file are stored in the file in the order in which they are written, and must be read back in the same order. Because of these characteristics, sequential files are often used for address books, dictionaries or other files which are searched from the beginning when they are used and relatively rarely updated.

The statements and functions used to write to or read from sequential files are as follows:

OPEN, CLOSE
PRINT #, PRINT # USING, WRITE #
INPUT #, LINE INPUT #
EOF, LOC

5.2.1 Creating sequential files

The steps involved in creating and accessing a sequential file are as follows:

(1) Execute an OPEN statement to assign a file number to the file and to open it in the "O" (output) mode.
(2) Write data to the file using the PRINT# or WRITE# statement.
(3) Close the file by executing a CLOSE statement. This must be done before the file can be reopened in the "I" mode for input.

Example:
The following is a short program which creates a sequential file of employee data. The file it creates will be used with subseauent programs. To simplify programming, type in all the data with the CAPS LOCK key set to SHIFT ON.

```
10 OPEN "O",#1,"A:EMPLOYEE.DAT":'Open file for output.
20 INPUT "NAME";N$              :'Assign name to N$.
30 IF N$="XX" THEN CLOSE:END    :'If XX is typed, the
40 '                              program ends.
50 INPUT "SECTION";S$           :'Assign section to S$.
60 INPUT "DATE OF BIRTH";D$     :'Assign date of birth to D$
70 PRINT #1,N$;",";S$;",";D$    :'Write data to file.
80 PRINT:GOTO 20                :'Moves cursor down.
```

Try running the program. The following prompt appears on the LCD screen:

NAME?

Type a name of employee (e.g. JOE SOAP) and press the RETURN key: the following message appears:

SECTION?

Type the name of section where the employee works (e.g. ACCOUNTS) and press the RETURN key: the following message appears:

DATE OF BIRTH?

Type the date of birth of the employee (e.g. 08/05/49) and press the RETURN key: the first message appears again. The above steps are repeated until "XX" is typed in response to the message "NAME?".

Continue in this manner so that the following items are input to the file.

NAME	SECTION	DATE OF BIRTH
JOE SOAP	ACCOUNTS	08/05/49
FRED BLOGGS	ENTERTAINMENT	01/02/60
BETTY JONES	ACCOUNTS	09/09/55
GLORIA SMITH	CUSTOMER SERVICE	03/04/62

Note that starting the program again after it has been terminated (that is, after the file has been closed) erases all the data entered previously and creates a new file to which the same name has been assigned.

After the program has been executed, if the command FILES "A: is typed a file "EMPLOYEE.DAT" should have been added to the directory.

5.2.2 Accessing sequential files

The procedures for accessing sequential files are as follows:

(1) Execute an OPEN statement to open the file in the "I" mode.
(2) Read data from the file into variables in memory by executing either the INPUT# or the LINE INPUT# statement.
(3) Close the file after input has been completed by executing a CLOSE statement.

Notes:
1) Data is read from the beginning of the file each time the file is opened.
2) If all data included in the file is to be read at once, a DIM statement must be executed to dimension one or more variable arrays of the appropriate size.
3) An "Input past end" error will occur if an attempt is made to read data from a sequential file after the end of that file has been reached.

Example:
The following program reads data from the file created by the sample program in Section 5.2.1 and displays the names of all employees working in the ACCOUNTS section.

```
10 OPEN"I",#1,"A:EMPLOYEE.DAT"       :'Open file for input.
20 IF EOF(1) THEN GOTO 110           :'If EOF is encountered,
30 '                                   file closed and program
40 '                                   ends.
50 INPUT#1,N$,S$,D$                  :'Read data and assign
60 '                                   items to N$,S$ and D$
70 IF S$="ACCOUNTS" THEN PRINT N$ :'When the section is
80 '                                   ACCOUNTS, name is
90 '                                   displayed on screen.
100 GOTO 20                          :'Next data record
110 CLOSE:END                        :'Close file and end
120 '                                  program
```

The screen will show the result as follows:

```
RUN
JOE SOAP
BETTY JONES
```

The comparison for the "ACCOUNTS" string requires the string not only to be spelt the same, but that it be all in upper case. This was why the suggestion was made to set the CAPS LOCK in the program to write the data, otherwise the program would have had to check for all possibilities, for instance "Accounts", "accounts", etc.

5.2.3 Updating sequential files.

After a sequential file has been written to a disk or microcassette, it is not possible to add data to that file once it has been closed. The reason for this is that the contents of a sequential disk file are destroyed whenever that file is opened in the "O" mode. To overcome this, the following procedures can be used:

(1) Open the existing file in the "I" mode.
(2) Open a second file on the disk or microcassette in the "O" mode under a different file name.
(3) Read in data from the original file and write it to the second file, adding the new data.

(4) After all data included in the original file has been written to the second file, close the original file and delete it with the KILL command.
(5) Write the new information to the second file.
(6) Rename the second file using the name which was assigned to the original file, then close the file.

The result is a sequential file which has the same file name as the original file, and which includes both the original data and the new data. A sample program illustrating this technique is shown below.

```
10 ON ERROR GOTO 310                    :'If file not found,
20 '                                      jump to 310.
30 '                                      If file exists, write
40 '                                      it to A:TEMP.
50 OPEN "I",#1,"A:EMPLOYEE.DAT"         :'Open file EMPLOYEE.
60 '                                      DAT for input.
70 OPEN "O",#2,"A:TEMP"                 :'Open temporary file
80 '                                      A:TEMP for output.
90 IF EOF(1) THEN 180                    :'If EOF is encountered
100 '                                      jump to 180.
110 LINE INPUT#1,A$                      :'Read data into A$.
150 PRINT#2,A$                           :'Write data in A$ to
160 '                                      A:TEMP.
170 GOTO 90                              :'Next data
180 CLOSE#1                              :'After all data has
190 '                                      been read, original
200 '                                     file is closed.
210 KILL "A:EMPLOYEE.DAT"               :'Kill original file
220 INPUT "NAME";N$                     :'Add new file entries.
230 IF N$="XX" THEN 280                 :'If XX is typed, close
240 LINE INPUT "SECTION? ";S$           :'file and end program.
250 LINE INPUT "DATE OF BIRTH? ";D$ :'
260 PRINT#2,N$;",";S$;",";D$            :'Write data to A:TEMP
270 PRINT:GOTO 220                      :'Next data
280 CLOSE                               :'Close A:TEMP.
290 NAME "A:TEMP" AS "A:EMPLOYEE.DAT":'Change filename back
300 '                                   to "EMPLOYEE.DAT"
310 IF ERR=53 AND ERL=30 THEN PRINT "File not found"
320 '                                     If A:EMPLOYEE.DAT not
330 '                                     exists, display "File
340 '                                     not found".
350 CLOSE:END                           :'
```

The contents of the original file could be changed by replacing the contents of variables before writing them to the second file. This could be done by adding the following sequence between lines 110 and 150.

```
111 PRINT A$                         :'Display contents of A$.
112 PRINT "Change entry(Y/N)?"
113 YN$=INPUT$(1)                    :'Wait for character input.
114 IF YN$="Y" THEN 117              :'Go to 117 if Y typed.
115 IF YN$="N" THEN 150              :'Go to 150 if N typed.
116 GOTO 112                         :'Go to 112.
117 INPUT "Enter new name";NN$       :'Input new entries.
118 INPUT "Enter new section";SS$:'
119 INPUT "Enter new birthdate";DD$
120 A$=NN$+","+DD$+","+DD$           :'Assign new entries.
```

5.3 Random Files

More program steps are required to create and access random files than is the case with sequential files; however, random files have two advantages which make them more useful when there are large quantities of data which must be frequently updated. The first is that random files require less disk space for storage because data is recorded using a packed binary format, whereas sequential files are written as series of ASCII characters. The second advantage is that random files allow data to be accessed anywhere on the disk; it is not necessary to read through each data item in sequence, as is the case with sequential files. Random access is made possible by storing and accessing data in distinct, numbered units called records.

The statements and functions which are used with random files are as follows.

OPEN, CLOSE
FIELD, LSET/RSET
GET, PUT
LOC, LOF
MKI$, CVI
MKS$, CVS
MKD$, CVD

5.3.1 Creating random access files

The steps required to create random files are as follows:

(1) Open the file in the "R" mode.

For example

OPEN "R", #2, "STOCKLST.DAT", 50

opens file number 2 as a random access file named "STOCKLST.DAT". The record length is 50 bytes. If the record length is omitted, records of 128 bytes are assumed.

(2) Next, allocate space in the file buffer for each of the variables which are to be written to the random access file. This is done using the FIELD command, for example:

FIELD #2 , 1Ø AS S$, 3Ø AS N$, 1Ø AS C$

This allocates the fields for file number 2. The example distributes the total length of the record among the variables as follows:

1Ø bytes for S$, 3Ø₃ for N$, and 1Ø for C$

Note that all fields are allocated as strings. Even numeric variables are stored as strings and must be converted as shown in step (3). Make sure the total equals the declared record length. If this length is exceeded, a

FIELD overflow in <line number>

error will be generated.

(3) Data is then placed into the random file buffer using the LSET and RSET commands, depending on whether they require left or right justification. Only strings can be placed into the buffer so any numeric values must be converted to strings first; this is done using the MKI$, MKS$, and MKD$ functions. For example:

LSET S$ = MKI$(S%)	converts and sets an integer.
LSET N$ = MKS$(Q!)	converts a single precision number.
LSET R$ = MKD$(R#)	converts a double precision number.
LSET N$ = A$	sets a string.

(4) Write data to the file from the random file buffer with the PUT statement. Records can be written in any order (in contrast to sequential files) and thus to add or change records it is only necessary to reopen the file and write further records. HOWEVER, with MICROCASSETTE files, the files MUST be written in sequential order.

(5) When all the data has been written to the file, the file must be closed using the CLOSE command:

e.g. **CLOSE**	closes all files.
CLOSE #2	simply closes file #2.

Failure to close a file may render the file impossible to be read from or written to at a later date.

The following program example allows data to be input from the keyboard for storage in a random access file. In this example, one record is written to the file output buffer each time the PUT statement on line 100 is executed. The record number which is used by the PUT statement is that which is input at line 30.

```
10 OPEN"R",#1,"A:STOCKLST.DAT",36
                    :'Open file                      -(1)
20 FIELD#1,2 AS S$,30 AS N$,4 AS C$
                    :'FIELD data to variables        -(2)
30 INPUT "ENTER STOCK NO.";S%
                    :'Input data items.
40 IF S%=0 THEN CLOSE:PRINT"END":END
                    :'Enter 0 to finish.
50 INPUT "ENTER ITEM NAME";A$
60 INPUT "QUANTITY";C%
70 LSET S$=MKI$(S%)
                    :'LSET data to buffer(file buffer -(3)
80 LSET N$=A$
90 LSET C$=MKS$(C%)
100 PUT#1,S%        :'Write data to file.             -(4)
110 GOTO 30         :'Next entry
```

NOTE:
Once a variable name is specified in a FIELD command, do not use that name in an INPUT or LET statement. The FIELD statement assigns variable names to specific positions in the random file buffer, and using an INPUT or LET statement to store values in a variable specified in the FIELD statement will cancel this assignment and reassign the name to normal string space instead of to the random file buffer.

5.3.2 Accessing random files

The following steps are required to retrieve data from a random access file:

(1) Open the file in the "R" mode.

(2) Using the FIELD statement, allocate space in the random file buffer for variables which are to be read in from the random file.

Note:
If the same program both writes data to a file and reads data from it, it is often possible to use just one OPEN statement and one FIELD statement.

(3) Move the desired record into the random file buffer with the GET statement. Any record number can be accessed without reading the whole file into memory as is the case with sequential files. HOWEVER, with microcassette files, the data MUST be accessed sequentially.

(4) Data in the random file buffer can now be used by the program. Be sure that numbers which are converted to ASCII strings for storage in the file are converted back into numeric values for use by the program; that is done using the CVI, CVS and CVD functions.

The following sample program accesses random file "STOCKLST.DAT" created using the program example shown in paragraph 5.3.1 above. Data records are read in and displayed by entering the stock number (record number) from the keyboard.

```
10 OPEN "R",#1,"A:STOCKLST.DAT",36
                     :'Open file in R mode.          -- (1)
20 FIELD#1,2 AS S$,30 AS N$,4 AS C$
                     :'Allocate space for variables. -- (2)
30 INPUT "ENTER STOCK NO.";S%
                     :'Input stock No.
40 IF S%=0 THEN CLOSE:PRINT"END":END
                     :'End program.
50 GET#1,S%
                     :'Read record into buffer.      -- (3)
60 PRINT USING "###";CVI(S$);:PRINT"     ";
                     :'Convert strings to numeric values
                       and display them.
70 PRINT USING "&";N$;:PRINT"     ";
80 PRINT USING "#####";CVS(C$)
90 GOTO 30
                     :'Next record
```

With random files, the LOC function returns the current record number; that is, the record number which is one greater than the number of the record last accessed by a GET or PUT statement. This function can be used to control the flow of program execution according to the total number of records which have been written to the file. For example, the following statement ends program execution if the current record number for file # 1 is greater than 50:

IF LOC(1) > 50 THEN END

5.3.3 Hints for increased performance

When BASIC is started, memory is automatically reserved for use as random file buffers. The amount of memory reserved equals the number of bytes specified in the /S: option (the maximum record length of random files) times the number of files specified in the /F: option (the maximum number of files which can be opened at one time). Specify 0 in the /S: option to conserve memory if random access files are not to be used. Also, specify /F: <number of files> in the BASIC command if fewer than three files (the default value) are to be used.

5.4 Microcassettes

The microcassette drive is supported as a disk device, and can be used in generally the same manner as a disk drive. It is, however, intended as a storage device which adds to the portability of the PX-8, and not as a device which would substitute for a disk drive in normal day to day use. It is obviously slower than a disk drive and has some further limitations. This section summarises the differences between the use of the microcassette as a disk drive and that of a conventional disk drive.

5.4.1 Restrictions on use

Since the microcassette is essentially a sequential access device, there are a number of restrictions on its use as a disk device.

(a) Only one microcassette file can be opened at once. If two files are opened in the input mode, only the second one to be opened can be accessed. When a file is opened in the output mode no other file can be opened until it has been closed.
(b) When a microcassette file is opened in the random ("R") mode it can be either read from or written to, but not both.
(c) When the GET statement is executed during random access, records must be read in sequence starting with record number 1. True random access is not possible.
(d) As with the GET statement, records must be accessed in sequence starting with record number 1 when the PUT statement is executed in the random mode. Further, the file must be one which has been opened for the first time; it is not possible to write data to a file which was previously created in the random mode. If an attempt is made to write another record to a file previously stored on tape a "Tape access error" will be generated.

5.4.2 Opening options

The options for opening a file with the microcassette are the same as for any other file with two additions. The full syntax is:

```
OPEN " |O| ", #n, "h:(sv)filename.ext"
       |I|
       |R|
```

The "s" option specifies whether the stop mode or the non-stop mode is to be used for reading from and writing to the file as follows:

S — Stop mode
N— Non-stop mode

Data is saved to the tape in blocks of 256 bytes. These blocks are duplicated on the tape. In order to achieve greater accuracy in reading and writing data these options allow the tape to be stopped between writing the duplicated blocks. In a normal BASIC SAVE operation data is always written in non-stop mode. It is possible to LOAD files in the stop mode, for instance:

```
LOAD "H:(S)PROGRAM"
```

will load the program named "PROGRAM" from the tape, stopping between each block. The tape will physically stop and start during this operation. It takes longer than normal operation and should only be used for loading BASIC programs if difficulty is experienced in loading a particular program.

In opening a data file for input it is often possible for a "Disk read" error to occur if the file was originally written in the non-stop mode and is read in the same mode. Greater reliability can thus be achieved by using the stop mode for reading back data.

If the option is omitted when the file is opened in the "I" mode, the mode actually used is that specified when the file was created. When a new file is created in the "R" mode, data is written in the stop mode unless otherwise specified.

The "v" option specifies whether data is to be automatically verified after being written, as follows:

V — Data is automatically verified (with a Cyclic Redundancy Check) from the beginning of the file through to its close after write access has been completed.
N— Data is not verified.

When omitted, the value set in the system display is assumed.

5.5 Errors

5.5.1 Error messages and causes

(1) Disk read error
An error occurred while data was being read from a disk.

(2) Disk write error
An error occurred while data was being written to a disk.

(3) Device unavailable
Access was attempted to a drive which did not contain a diskette, or the specified drive was not connected.

(4) Disk write protected
An attempt was made to write data to a disk which was protected with a write protect tab.
An attempt was made to write data without executing the RESET command after the diskette in that drive had been replaced.
An attempt was made to write data to a file for which the write protect attribute was set.
An attempt was made to write data to a ROM device.

(5) Tape access error
An attempt was made to access an access-inhibited microcassette file.
An attempt was made to MOUNT a tape without REMOVEing the previous tape.
An attempt was made to REMOVE a tape which has not been MOUNTed.

5.5.2 Error processing

(1) Errors occurring when a file is opened
Identify and eliminate the cause of the error, then re-execute the OPEN statement.

(2) Errors occurring during output
CLOSE the applicable file immediately if any of the following errors occur during output with statements such as PRINT # or PUT. The contents of the file may be destroyed if output is continued.

Device unavailable
Disk write protected
Disk read error
Disk write error

(3) Errors occuring when a file is closed
Although a file will be closed if an error occurs when a CLOSE statement is executed, the contents of the file are not assured. Further, if an error occurs when an attempt is made to close more than one file with a single CLOSE statement there is a possibility that some files will not be closed. If an error occurs in this situation, additional CLOSE statements must be executed until no further errors occur.

(4) "Disk write protected" error
This error will occur if a disk is changed without having CLOSEd the files on the original disk. All files should be closed before a disk is changed, then the RESET command should be executed.
If the disk is changed without closing the files, the RESET command will close the files. However, it will be necessary to execute the RESET command a number of times until no further errors occur before the new disk can be accessed.

5.6 Precautions On Changing Floppy Disks

This section may be skipped if you do not use an optional floppy disk drive unit.

Before removing a floppy disk from the drive, be sure to CLOSE all files currently open on that drive. The reason for this is as follows:

Assume that a file on the disk being replaced is open in the "O" or "R" mode and that data has been output to that file with the PRINT # or PUT statements. Write operations to the disk by these statements are not necessarily actually made until the file is closed. Therefore, if the floppy disk is replaced without executing the CLOSE statement the contents of the file on that disk are not assured. Further, if another disk is inserted in place of the one on which the file was opened, the contents of the disk on that drive may be destroyed when an attempt is made to CLOSE the file.

To avoid the destruction of data to the maximum extent possible, the CP/M operating system is designed so that disks are automatically write protected on replacement. If an attempt is made to write data to a disk while it is this condition a "Disk write protected" error will occur. The write protected condition can be cleared and write access to the new floppy disk enabled by executing the RESET command.

For these reasons, the following procedures should be observed when replacing floppy disks:

In the direct mode:

```
        CLOSE all files;
        Replace disk;
        Execute RESET.
```

During program execution:

```
        100 CLOSE
        110 PRINT "Change the disk!"
        (Change the disk and press any key)
        120 A$ = INPUT$(1)
        130 RESET
```

Chapter 6

SEQUENTIAL ACCESS USING DEVICE FILES

This Chapter describes procedures for sequential access to the RS-232C serial communications interface and other external I/O devices such as the keyboard, screen and printer.

6.1 Using the RS-232C Interface

The PX-8's RS-232C interface makes it possible to connect the PX-8 to RS-232C compatible devices such as printers, acoustic couplers, or other PX-8s. Support for RS-232C interface access is a standard feature of BASIC for the PX-8. The RS-232C port is handled as a sequential input/output device, and is identified by the device name "COMØ:".

6.1.1 Opening the RS-232C interface

The RS-232C communications interface is opened for data communication by executing an OPEN statement. The parameters of this statement specify the mode in which the interface is to be opened (input or output), the file number whicl: is to be assigned to the device and the communication protocol and control options. The communication protocol and control options are specified as a string of up to seven characters, each of which determines the setting of one of seven communication options. Devices which are connected to the RS-232C port must be compatible with the communication protocol under which the interface is opened.

The format for specification of the OPEN statement and the meanings of the various communication options are described in detail below.

(1) OPEN statement

The general format of the OPEN statement for opening the RS-232C interface port is the same as that used when opening sequential access files on floppy disks or in the RAM disk. However, the format of the file descriptor differs slightly.

For disk files the format allows an optional device name but the file must be given a name, for example:

OPEN "O", # 3,"A: TEST FILE"
OPEN "O", # 1,"NAME"

For the RS-232C interface port the format is as follows:

OPEN "<mode>", # <file number>, "COMØ: [(<options>)]"

"COMØ:" must always be specified when opening the port, and no file name is required. However, the options for determining the communication mode and protocol need not be specified; if they are omitted, the default values of CP/M are used. After the PX-8 is initialized, these values are as follows; they remain effective until changed with the CONFIG program of CP/M or the OPEN statement of BASIC.

Data transfer rate:	4800 bits per second
Word length (bits/character):	8
Parity:	None
No. of stop bits:	2
DSR send check:	OFF
DSR receive check:	OFF
DCD check:	OFF
SI/SO control:	OFF
XON/XOFF:	OFF

Opening an RS-232 port for output would thus take forms such as:

OPEN "O", # 3,"COMØ:"
OPEN "I", # 1,"COMØ:(68E3F)"
OPEN "O", # 2,"COMØ:(68E3AXN)"

• Options for Protocol and Control

The <options> specification in the OPEN statement determines the data communication protocol and the control options. These are specified as a character string of up to seven characters, each of which determines the setting of one option. The general format of the string specifying these options is (blpscxh), where "b" specifies the bit rate, "l" the word length, "p" the type of parity check to be made, and "s" the number of stop bits between characters. Option "c" specifies which of the interface's four control lines are to be checked when the interface is opened and during data communication. The "x" option specifies whether or not communication is to be controlled according to XON/XOFF protocol (that is, whether the PX-8 is to issue or respond to "wait until I catch up" requests during communication with the device on the other end of the RS-232C line), and the "h" option indicates whether Shift-In/Shift-Out sequences are to be used to indicate whether characters are upper case or lower case (applicable only when the data word length is 7 bits and when it is necessary to send characters whose codes are 128 or greater). These options are summarized in the table on page 6-4.

WARNING:
When using the XON/XOFF or Shift-In/Shift-Out options, make sure they are reset to off on exiting from the program. It is only possible to set them from BASIC and not from the CONFIG program of the CP/M utility ROM.

If care is not taken the XON/XOFF or SI/SO options may be set when they are not required, this can result in the following problem when machine code is being received. The SI, SO, XON or XOFF characters may be part of the machine code. If the receiving computer has either or both of the XON/XOFF or SI/SO options set, when these characters are received they will be intercepted by the RS-232C software and acted upon. This means they will be lost as part of the machine code file. Moreover, if an SO character is received, the data following will be changed. It will have the high bit set. This will further corrupt the file. Similarly XON/XOFF characters can be intercepted and cause the transmisssion to hang indefinitely.

When using communications programs other than BASIC ones, a warm start should be made before setting the RS-232C parameters using the CONFIG program. This ensures that the SI/SO and XON/XOFF parameters are set to be off.

blpscxh protocol format

b	**BIT RATE**
	0: 1200 send, 75 receive 1: 75 send, 1200 receive 2: 110 — 3: Not specifiable 4: 150 — 5: Not specifiable 6: 300 — 7: Not specifiable 8: 600 — 9: Not specifiable A: 1200 — B: Not specifiable C: 2400 — D: 4800 E: 9600 — F: 19200
l	**WORD LENGTH**
	6: 6 bits 7: 7 bits 8: 8 bits
p	**PARITY**
	N: None E: Even O: Odd
s	**STOP BITS**
	1: 1 stop bit 2: 1.5 stop bits 3: 2 stop bits
c	**ACTIVE CONTROL LINES**
x	**XON/XOFF**
	X: On N: Off
h	**SHIFT-IN/SHIFT-OUT**
	S: On N: Off

ACTIVE CONTROL LINES (c):

Value	DSR send check	DSR receive check	DCD check
0 or 8	ON	ON	ON
1 or 9	ON	ON	OFF
2 or A	ON	OFF	ON
3 or B	ON	OFF	OFF
4 or C	OFF	ON	ON
5 or D	OFF	ON	OFF
6 or E	OFF	OFF	ON
7 or F	OFF	OFF	OFF

Meanings of each position in the "blpscxh" string are as follows.

b — A hexadecimal integer from &H0 to &HF which determines the bit rate as follows:

0 —	Send bit rate = 1200 bps, receive bit rate = 75 bps
1 —	Send bit rate = 75 bps, receive bit rate = 1200 bps
2 —	110 bps
3 —	Not specifiable
4 —	150 bps
5 —	Not specifiable
6 —	300 bps
7 —	Not specifiable
8 —	600 bps
9 —	Not specifiable
A—	1200 bps
B—	Not specifiable
C—	2400 bps
D—	4800 bps
E—	9600 bps
F—	19200 bps

l — A number from 6 to 8 which determines the number of bits per character (the data word length):

6:	6 bits/character
7:	7 bits/character
8:	8 bits/character

p — A letter which determines the type of parity check to be made:

N:	No parity check
E:	Even parity
O:	Odd parity

s — A number which determines the number of stop bits to be included between each character:

1:	1 bit
2:	1.5 bits
3:	2 bits

c — A hexadecimal digit from 00H to 0FH which determines which of the four control lines are checked. Correspondence between the settings of each of the four bits and the control lines to be checked is as follows:

Bit 3 — No meaning

Bit 2 — Data Set Ready (DSR) send check
1: OFF
0: ON

Bit 1 — Data Set Ready (DSR) receive check
1: OFF
0: ON

Bit 0 — Data Carrier Detect (DCD) check
1: OFF
0: ON

Combinations of settings for each hexadecimal digit are as follows:

	DSR send check	DSR receive check	DCD check
0 or 8	ON	ON	ON
1 or 9	ON	ON	OFF
2 or A	ON	OFF	ON
3 or B	ON	OFF	OFF
4 or C	OFF	ON	ON
5 or D	OFF	ON	OFF
6 or E	OFF	OFF	ON
7 or F	OFF	OFF	OFF

x — A letter which determines whether XON/XOFF (send ON/send OFF) protocol is to be used for communication control. When XON is specified and the interface is opened in the "I" mode, the PX-8 automatically outputs control code 19 (13H) during output via the RS-232C interface in the "O" mode, and automatically interrupts output until control code 17 (11H) is received from the device at the other end of

the line. This prevents data from being lost due to receive buffer overflow when the speed of data transmission is greater than the speed with which data received can be unloaded from the buffer.

X:	XON/XOFF protocol used for send control.
N:	XON/XOFF protocol not used for send control.

h — A letter which determines whether the shift-in/shift-out (SI/SO) control sequences are to be used. The SI/SO control sequences are used when sending 8-bit data with a data word length of 7 bits. Shift-in/shift-out control can be used only when the data word length is 7 bits:

S:	Shift-in/shift-out control used.
N:	Shift-in/shift-out control not used.

The (blpscxh) options can be completely or partially omitted when the communications interface is opened; however, spaces must be specified for options which are omitted if there are any following options. If b, l, p or s are omitted, the default values are those which have been set with the CONFIG command. If c is omitted, "F" is used, and if x and/or h are omitted, "N" is used.

(2) OPEN modes

The RS-232C interface can be opened in either the "I" or "O" modes. The "I" mode is specified for input and the "O" mode is specified for output. If both input and output are to be performed simultaneously, the interface must be opened as two files (one for input and one for output). In this case, the communication protocol and control options used are those specified in the first OPEN statement executed: the options will be ignored if they are included in the file descriptor of the second OPEN statement.

Example

```
10 OPEN"I",#1,"COM0:(68N3FXN)"
20 OPEN"O",#2,"COM0:(48E2F)"
.
.
.
100 PRINT#2,A$
110 INPUT#1,B
```

In the example above, the option specification (48E2F) on line 20 is ignored and the output file (#2) is opened using the protocol specified on line 10 (68N3FXN).

(3) Control lines used for communication through the RS-232C interface

(a) DTR (Data Terminal Ready)
DTR is a signal which is output by the PX-8 to indicate that it is ready for data communications. The level on this line becomes HIGH when the communications interface is opened in either the "I" or "O" mode, and becomes LOW when the interface is closed (when it is no longer open in any mode).

(b) RTS (Request To Send)
RTS is a signal which controls operation of a communication device (modem or acoustic coupler) connected to the PX-8. The signal on this line becomes HIGH when the interface is opened in the "O" mode, and LOW when the interface is closed.

(c) DSR (Data Set Ready)
DSR is a signal which indicates whether the communication device connected to the RS-232C port is ready for operation. When HIGH, the device connected to the interface port is ready to accept signals controlling data transmission/reception. When the interface is opened in the "I" mode with the DSR receive check bit (bit 1 of option "c") set to "0" (ON), OPEN statement execution is not completed until the level on the DSR line becomes HIGH.

(d) DCD (Data Carrier Detect)
This line is used for detecting the data carrier signal from the device connected to the RS-232C port. When the interface is opened in the "I" mode with the DCD check bit set to "0" (ON), the OPEN statement is not completed until the level on the DCD line becomes HIGH.

6.1.2 Output to the RS-232C Interface

Statements and functions used for access to the RS-232C interface are as follows:

Statements

OPEN, CLOSE, INPUT #, LINE INPUT #, PRINT #, PRINT # USING, LOAD, LIST, RUN, MERGE

Functions

EOF, LOC, LOF, INPUT$

The following statements can be used to output data to the RS-232C port.

PRINT #
PRINT # USING

When data is output using these statements, the data format is the same as when data is output to a disk drive.

(1) Control line checks for the "O" mode

(a) CTS (Clear To Send)
Output to the RS-232C port becomes possible when the level on this line becomes HIGH.

(b) DSR (Data Set Ready)
When the DSR send check bit is OFF (when bit 2 of option "c" is 1), data is output to the RS-232C port regardless of the level on the DSR line. When the DSR check bit is ON ("0"), data is output after checking the level on the DSR line and waiting for it to become HIGH.

(2) Errors applicable to the "O" mode

(a) Device unavailable
The RS-232C interface cannot be used.

(b) Device time out
The level on the DSR line did not become HIGH within a certain period of time when output to the RS-232C port was attempted after opening it in the "O" mode with the DSR send check bit (bit 2 of option "c") set to ON ("0"). This error also occurs if the STOP key is pressed while transmission is being deferred for some reason.

6.1.3 Input from the RS-232C interface

The following statements and functions are used to input data via the RS-232C interface.

Statements	Functions
INPUT #	**INPUT$**
LINE INPUT #	

The format in which data is input from the interface by these statements is exactly the same as in the case of input from disk files.

The INPUT # and LINE INPUT # statements do not allow full freedom of data format during input because they require pre-determined delimiters and termination symbols. However, the INPUT$ function permits input without regard for delimiters or terminators; thus it can be used with functions such as EOF and LOF to provide full freedom of format.

(1) Control line checks for the "I" mode

(a) DSR (Data Set Ready)
If the DSR receive check bit is set to ON (if bit 1 of option "c" is set to 0), the DSR line is monitored during input and an error is generated if it drops to LOW.

(b) DCD (Data Carrier Detect)
When the interface is opened for input with the DCD check bit set to ON (with bit 0 of option "c" set to 0), the level of the DCD line is checked at the time of execution of an OPEN "I" statement for the RS-232C interface and the port is not opened until the DCD line becomes HIGH. An error is generated if the level on this line becomes LOW during input.

(2) Errors applicable to the "I" mode

(a) Device unavailable
This error occurs when the RS-232C interface cannot be used for some reason.

(b) Device time out
This error occurs if the level on the DSR line does not become HIGH within a certain period of time after an attempt is made to open the RS-232C interface in the "I" mode with the DSR receive check bit set to ON (with bit 1 of option "c" set to 0). The same is true if the level on the DCD line does not become HIGH within a certain period of time after an OPEN"I" statement is executed with the DCD check bit set to ON (with bit 0 of option "c" set to 0)

(c) Device fault
This error occurs if the RS-232C port is opened for input with the DSR receive or DCD check bits set to ON (with bits 1 or 0 of option "c" set to 0), and the level of a corresponding line becomes LOW during input.

(d) Device I/O error
This error occurs if a parity error, overrun error or framing error occurs during input. Although this error is reset if input is continued, there is no assurance that data received into the receive buffer at that time will be correct.

(e) Input past end
This error occurs if the STOP key is pressed during input from the RS-232C interface with INPUT #, LINE INPUT # or INPUT$.

6.1.4 RS-232C functions

The four functions used with the RS-232C interface are as follows.

EOF
LOC
LOF
INPUT$

(1) EOF (<file no.>)
This function returns −1 (true) when the receive buffer is empty, and 0 (false) when the buffer is not empty.

(2) LOC (<file no.>)
This function returns the number of bytes of data remaining in the receive buffer.

(3) LOF (<file no.>)
This function returns the number of free bytes remaining in the receive buffer.

NOTE:
The size of the receive buffer is 262 bytes.

(4) INPUT$(<no. of characters>,<file no.>)
This function inputs the specified <no. of characters> from the RS-232C interface and returns them as a character string.

6.1.5 Using the LOAD, SAVE and LIST commands with the RS-232C interface

Programs can be output via the RS-232C interface in ASCII format by using LIST "COMØ:",A. When this is done, CTRL-Z (code 26, an end mark) is output after transmission of the program has been completed. However, when SAVE "COMØ:" is executed, the program saved is output in ASCII format regardless of whether the A option or the P option is specified.

When an ASCII program is loaded via the RS-232C interface with LOAD "COMØ:", loading is terminated when CTRL-Z is received. The same applies when the program is loaded using RUN "COMØ:".

If CTRL-Z is not received after receiving a program through the RS-232C interface, loading can be terminated by pressing CTRL together with the STOP key.

NOTE:
When transferring BASIC programs via the RS-232C interface, either specify a data word length of 8 bits or use Shift-in/Shift out control with a word length of 7 bits.

6.2 Printer and Display Screen

With PX-8 BASIC, a printer and the LCD screen are supported as sequential output devices. This means that data can be output to the printer or screen using the file output statements (PRINT # and PRINT # USING), as well as the dedicated printer/display statements LIST/LLIST and PRINT/LPRINT.

The device name used to open the printer as a device file is "LPTØ:", and that used to open the display screen is "SCRN:".

(1) Statements
Statements which can be used for output to the display screen or printer when it is handled as a device file are as follows:

OPEN, PRINT #, PRINT # USING, CLOSE, LIST "<file descriptor>"

(2) Errors
The "Device time out" error will occur if the printer is not ready for output (because it is offline or out of paper, for instance).

6.3 Keyboard

PX-8 BASIC also allows the keyboard to be handled as a sequential access input device. When the keyboard is opened as a file, data input is assigned to variables using INPUT#, LINE INPUT# and INPUT$ (X,<file no.>) instead of the corresponding dedicated keyboard input statements. This makes it possible to use common routines for input of data from the keyboard, disk device files and the RS-232C interface.

The device name used to OPEN the keyboard as a device file is "KYBD:".

(1) Statements
Statements which can be used for input from the keyboard when it is handled as a device file are as follows:

CLOSE, INPUT#, INPUT$ (X,<file no.>),
LINE INPUT#, LOAD, OPEN "I"

(2) Errors
A "Bad file descriptor" error will occur if an attempt is made to open the keyboard in the "O" mode.

Appendix A ERROR CODES AND ERROR MESSAGES

When an error occurs in a BASIC program, it is detected by the interpreter and a message is printed. In most cases the error stops the program and will not allow it to continue. BASIC will return to the direct mode and present the error message. It will not always be obvious what exactly has caused the error. It may be something as simple as a mistyped command which BASIC does not recognise, an error of logic or any one of a series of programming faults. This appendix is an attempt to help the user/programmer to find out what exactly he has done wrong. It is not easy to cover each and every cause of an error, because some errors are particular to the logic of a program and simply cannot be predicted. However, many are due to definite reasons, and these are described below.

Each error has a code associated with it, which is useful for trapping errors and also simulating them. See ERROR, ON..ERROR, ERR and ERL in Chapter 4 for details of their use. A list of errors in numerical order is given at the end of this section. However, as the error is normally encountered as a message, the details of each error are given in alphabetical order. The number at the left of each error is the error code.

54 Bad file mode

A statement or function was used with a file of the wrong type.

Possible causes:
(i) An attempt was made to use PUT, GET or LOF with a sequential file.
(ii) A non BASIC program file was specified in a LOAD command.
(iii) A file mode other than I, O, or R was specified in an OPEN statement.
(iv) An attempt was made to MERGE a file that was not saved in ASCII format.

64 Bad file descriptor

An illegal file name was specified in a LOAD, SAVE or KILL command or an OPEN statement (for example, a file name with too many characters).

52 Bad file number

A statement or command references a file that has not been opened, or the file number specified in an OPEN statement is outside of the range of file numbers that was specified when BASIC was started.