GPL INTERFACE SPECIFICATIONS FOR THE 99/4 DISK PERIPHERIAL

CONSUMER GROUP
MAIL STATION 5890
2301 N. UNIVERSITY
LUBBOCK, TEXAS 79414

March 28, 1983

Contents

1 Introduction **5**

2 APPLICABLE DOCUMENTS **7**

3 DISK DSR LEVEL CONCEPT **9**

3.1 Level 1 Subroutines 9

3.1.1 Sector READ/WRITE - SUBPROGRAM 010 10

3.1.2 Disk Formating - SUBPROGRAM 011. 10

3.2 Level 2 Subroutines 11

3.2.1 Modify File Protection - SUBPROGRAM 12 11

3.2.2 File Rename Routine - SUBPROGRAM 013 12

3.3 Direct File Access Routines 12

3.3.1 Access Direct Input Files - SUBPROGRAM 014 12

3.3.2 Access Direct Output File - SUBPROGRAM 015 13

3.4 Buffer Allocation Routine - SUPROGRAM 016 14

List of Tables

3.1 Additional Information Block 13

3.2 Additional Information Block 14

Chapter 1

Introduction

The information contained in this document gives a complete specification of the interface between the 99/4 Disk Peripheral and the GPL interpreter.

NOTE

Throughout this document hexadecimal numbers are indicated by a preceeding 0 or a preceeding >. Therefor the numbers 010, >10, and decimal 16 are the same value.

The items in the transfer blocks which are enclosed braces {} are items returned by the sub-program.

Chapter 2

APPLICABLE DOCUMENTS

- File Management Specification for the 99/4 Home Computer (version 2.5, Revised 25 February 1983)
- Home Computer BASIC Language Specification (Revision 4.1, 12 April 1979)
- Home Computer Disk Peripheral Hardware Specification
- Functional Specification for the 99/4 Disk Perpheral (Version 3.0, Revised 28 March 1983)
- GPL Interface Specification for the 99/4 Disk Perpheral (Version 2.0, revised 28 March 1983)

Chapter 3

DISK DSR LEVEL CONCEPT

The disk DSR has been developed as a 3 Level software package, each level defines distinct options that can be used on higher levels. This section will give a brief overview of the levels used and of the features built-in to each level.

The three levels used are:

LEVEL 1 Definition of the basic disk funtions like sector READ/WRITE. Head control, drive selection, and track formatting.

LEVEL 2 Definition of the "file" concept. Each file is addressable it's name and an offset of a 256-byte block relative to the beginning of the file.

LEVEL 3 Extension of the file concept to the level given in the file magement specifications. Introduction of the logical FIXED or VARIABLE length records, RELATIVE record or SEQUENTIAL files.

The following sections will each describe a level and the related subprogram calls to it.

3.1 Level 1 Subroutines

The lowest level routines in the DSR are called Level 1 Subroutines. These routines make the higher levels independent of the physical disk medium, e.g. changing the disk software for double density would only involve changing the subroutines on this level, as long as the physical sector size remains at 256 bytes.

The folowing routines are availble on this level:

- SECTOR READ/WRITE
- FORMAT DISK

The following sections will contain a description of these routines and thier call requirements. All parameters will be tranferred throught the FAC block in CPU RAM. This block is located in CPU RAM satarting at relative location 04A (currently >834A).

3.1.1 Sector READ/WRITE - SUBPROGRAM 010

The transfer block for this subprogram is:

004A	{ Sector Number }		004B
004C	Unit #	READ/WRITE	004D
004E	VDP Buffer start address		004F
0050	Sector Number		0051

ERROR CODES ARE RETURNED IN CPU LOCATION 050
The meaning of each entry is:

SECTOR NUMBER number of the sector to be written or read. Sectors are addressed as logical sectors (0-359 for a standard minifloppy) rather than a track and record number, which would require the knowledge of the physical layout of the floppy disk. The sector number has to be given in CPU RAM locations 050-051, and will be returned in CPU RAM locations 04A-04B.

UNIT # Indicates the disk drive that the operation is to be preformed. This entry has to be either 1,2, or 3.

READ/WRITE Indicates the direction of data flow.
0=WRITE
$\neq$ 0=READ

VDP BUFFER START ADDRESS Indicates the start of VDP data buffer for data transfer. The number of bytes transferred will always be 256.

ERROR CODES WILL BE RETURNED IN CPU LOCATION 050.

3.1.2 Disk Formating - SUBPROGRAM 011.

This routine will format the entire disk on a given unit unless the disk in the unit has been hardware write protected. It can use any VDP memory, starting at the location given in the transfer block. The amount of memory used depends on the disk format. For the current single density format, the buffer memory used is a nominal 3125 bytes. This can vary with a disk motor speed to a maximum of 3300 bytes. To be compatible with double density versions of the disk controller, the minimum buffer size must be 8K.

The transfer block for this subprogram is:

004A	{ # of sectors/disk }			004B
004C	DSR Ver	Unit #	# of tracks	004D
004E	VDP Buffer start address			004F
0050	Density		# of sides	0051

The meaning of each entry is:

# OF SECTORS/DISK Is returned by the routine to provide comatibility between the current controller version and future (double density or SA200) versions.

DSR VERSION This is the MSNibble

- **0** indicates the format requires nothing special and can be done by any version of the DSR.
- **1** indicates the format requires the 2nd version of the DSR for 1 of 2 reasons. It may be a double sided format is requested or it may be because a # of tracks other than 35 or 40 is requested.
- **2** indicates the format requires features that are not on the currrent DSRs. (Density and perhaps double tracking if it is available on the next DSR.)

UNIT # Indicates the disk drive the operation is to be preformed. This entry must be 1,2, or 3. This is the LSNibble

# OF TRACKS Indicates the number of tracks to be formated. In the current version this number must be 35 or 40!!! Upon return, this entry contains the number of sectors/tracks.

VDP BUFFER ADDRESS Indicates the start of the VDP buffer that can be used by the disk controler to write tracks.

DENSITY Only single is supported at this time.

# OF SIDES Indicates the number of sides to format.

3.2 Level 2 Subroutines

The Level 2 Subroutines are those that use the concept "file" rather than "logical record number". Notice that the file concept on this level is limited to an abstract type of file which has no properties such as "program file" or "data file". A file on this level is merely a collection of data, stored in logical blocks of 256 bytes each.

The logocal blocks on this level are accessed by file name and logical block offset. This offset starts block 0 and ends with block N-1 with a file length of N blocks.

3.2.1 Modify File Protection - SUBPROGRAM 12

The transfer block for this subprogram is:

004C	Unit #	Protect code	004D
004E	Pointer to the file name		004F

This protection bit for the indicated file will be set or reset according to the information given in CPU location 04D:

0= Reset the file protect bit. The file is no longer protected against modification/ deletion.

OFF= Set the file protection bit. Disallow SAVE and OPEN for OUTPUT, APPEND, or UPDATE mode.

3.2.2 File Rename Routine - SUBPROGRAM 013

The transfer block for this subprogram is:

004C	Unit #	Unused	004D
004E	Pointer to new name		004F
0050	Pointer to old name		0051

Both pointers, located at 04E and 050 in CPU RAM, point to the VDP location of the first character of the file name. The first pointer points to the new name, the second one to the original file name. Each name is left adjusted in a 10 character field, filled with spaces. Each name is located in VDP RAM and has to be a legal file name. No checks are being made to ensure legality of the name.

Since the rename has to be done on the same disk, only one unit # entry is required. This unit # is located in CPU RAM location 04C.

Error codes are returned at location CPU RAM 050. The error codes returned are identical to the standard file management error codes, i.e. only the upper three bits of the error byte are significant.

3.3 Direct File Access Routines

The direct file access routines can be used for accessing disk files without paying attention to the type of disk file (PROGRAM or DATA). The level of access is equivalent to the Level 2 disk software, which means that the access is performed on the basis of straight AUs. However, Level 3 information can be passed at file open time.

Since the input and output direct access subprograms can be used together to copy files, the user has to be careful with the information returned by the input file subprogram, since some of this information may be used by the output file subprogram.

3.3.1 Access Direct Input Files - SUBPROGRAM 014

The transfer block for this subprogram is:

004C	Unit #	Access Code	004D
004E	Pointer to File Name		004F
0050	Addt'l info		

The meaning of each entry is:

Unit # indicates the disk drive on which the operation is to be performed. Must be 1,2, or 3.

Access code an access code is used to indicate which function is to be performed, since this subprogram contains multiple functions. The following codes are used:

0→ Transfer file parameters. This will transfer Level 2 parameters to the additional information area (6 bytes). It also passes the number of AUs allocated for the file.

N→ When N is not equal to 0, this indicates the number of AUs to be read to a given file, starting at the AU indicated in the additional information block.

After the READ is complete, This entry contains the actual number of AUs read. If all AUs have been read, this entry will be 0.

Pointer to file name Contains a pointer to the first character of the 10 character filename., possibly padded to the right with spaces. This filename is NOT checked by the disk software.

Additional info Points to a 10 byte location in CPU RAM containing information for direct disk access:

Table 3.1: Additional Information Block

X	VDP Buffer Start Address	
X+2	# of first AU	
X+4	Status Flags	# records/AU
X+6	EOF offset	Log.Rec.Size
X+8	# of Level 3 records alloc.	

The VDP Buffer start address is where the information read from the disk is stored. The Buffer has to be able to store at least N×256 bytes, in which N is the access code.

The # of the first AU entry is the AU number at which the read should begin. If the access code =0 (parrameter passing), the total number of AUs allocated for the file wil be returned.

The remaining 6 bytes are explained in the Software Spec. for the 99/4 Disk Peripheral. The user must be careful when changing these bytes, since they directly affect Level 3 operation. If they are not modified consistently, weird things may happen.

ERROR CODES ARE RETURNED AT LOCATION 050 IN CPU RAM

3.3.2 Access Direct Output File - SUBPROGRAM 015

The transfer block for this subprogram is:

004C	Unit #	Access Code	004D
004E	Pointer to the file Name		004F
0050	Addt'l Info		

The meaning of each entry is:

Unit # is the drive being used, must be 1, 2, 3.

Access Code an access code is used to show which function is to be preformed, since multiple functions are combined, the following codes are used:

0→ Create file and copy Level 3 parameters from addt'l info area.

N→ If N $\neq$ 0, the number of AUs to be written to the file, starting at the AU in the Addt'l Info Block.

Pointer to file name Contains a pionter to the first character of a 10 character filename. (Maybe padded with 0's). The filename is NOT checked ny the disk software.

Additional Info Points to a 10 byte location in CPU RAM containing addt'l info for direct disk access:

Table 3.2: Additional Information Block

X	VDP Buffer Start Address	
X+2	# of first AU	
X+4	Status Flags	# records/AU
X+6	EOF offset	Log.Rec.Size
X+8	# of Level 3 records alloc.	

The VDP Buffer start address is where the information read from the disk can be stored. It must be able to store N×256 bytes, Where N is the access code.

The # of first AU entry is the AU number at which the read should begin. If it is =0 (parameter passing), the total number of AUs to be used must be indicated.

The remaining 6 bytes are explained in Software Specs. for the 99/4 Disk Peripheral.

ERROR CODES ARE RETURNED TO LOCATION 050 IN CPU RAM

3.4 Buffer Allocation Routine - SUPROGRAM 016

The argument for this subprogram is the number of file buffers to be used. This argument is given in FAC+2 (CPU location 04C).

The effect of this routine is that an attempt is made to allocate enough VDP space for the disk usage to facilitate the simultaneous opening of the given number of files. This number is between 1-16.

The disk software automatically relocates all buffers that have been linked in the following manner:

Byte 1 Validation code

Byte 2-3 Top of memory before allocation of this buffer.

Byte 4 High byte of CPU address for the given buffer area. For programs this byte is 0.

The linkage to the first buffer area is made through the current top of memory, given in CPU location 070 (currently >8370).

The top of memory is also automatically updated after successful completion of this subprogram.

A check is made that the current request leaves at least 0800 bytes of VDP space for the screen and data storage. If this is not the case, or if the total number of buffers requested is 0 or >16, the requst is ignored and an error code will be indicated in CPU location 050 (currenly >8350).

Successful completion is shown by a 0 byte in CPU location 050. A nonzero here means tuff luck.

FUNCTIONAL SPECIFICATIONS FOR THE 99/4 DISK PERIPHERAL

CONSUMER GROUP
MAIL STATION 5890
2301 N. UNIVERSITY
LUBBOCK, TEXAS 79414

MARCH 28, 1983

Chapter 1

INTRODUCTION

The information contained in this document is intended to give a complete functional specification of the 99/4 Disk Peripheral as seen from the Basic user stand point.

Chapter 2

APPLICABLE DOCUMENTS

- File Management Specification for the 99/4 Home Computer (Version 2.5, Revised 25 February 1983)
- Home Computer BASIC Language Specification (Revision 4.1, 12 April 1979)
- Home Computer Disk Peripheral Hardware Specification
- Functional Specification for the 99/4 Disk Peripheral (Version 3.0, Revised 28 March 1983)
- GPL Interface Specification for the 99/4 Disk Peripheral (Version 2.0, Revised 28 March 1983)

Chapter 3

SUPPORTED FILE MANAGEMENT OPTIONS

The disk peripheral supports most of the options in the File Management Spec. for the 99/4 Home Computer.

The supported options include:

- Sequential and Relative record (random access files)
- Fixed and Variable length records
- Internal and Display file types
- Output, Input, Update, and Append access modes
- Program Load and Save functions

The I/O routines supported by the disk peripheral are:

- OPEN - Open an existing file for access. This routine must have the drive number or the disk name and the filename to open.
- CLOSE - Close a file for access. The PAB can be released and the disk peripheral software deallocates some buffer area in VDP memory. Since the number of files tat can be open at once is limited it is advised each file is closed as soon as it is no longer needed.
- READ - Read a logical record from an open file.
- WRITE - Write a logical record to an open file.
- RESTORE/REWIND - relocate the file read/write pointer to a given location in the file. For sequential files this can only be the beginning of the file, whereas for relative record files, the file read/write pointer can be relocated to any logical record in the file by giving the record number.
- LOAD - Load a program file into VDP memory. The disk peripheral will check the correct file type before the program is loaded (see section 4.7).

- SAVE - Save a program in VDP memory onto the named disk file. The disk peripheral does not check for legal BASIC memory images, so this routine, like the LOAD routine, can be used for transferring binary memory data to and from disk files. Note that the disk file is marked as a program file however, so that files created with a SAVE command can only be read with a LOAD command.
- DELETE - Delete the indicated file from the given disk, delete frees up the space occupied by the file for future use.
- SCRATCH RECORD - This function is not supported by the disk peripheral.
- STATUS - Indicates current status of a file. This includes the logical and physical EOF flags and the protection flag.

Chapter 4

INTERFACE TO BASIC

This section will provide a general overview of how the disk peripheral presents itself to the BASIC user. For the BASIC related details the reader is referred to the Home Computer BASIC Language Specs.

4.1 OPEN Statement

The BASIC OPEN statement allows the user to access files stored on accessory devices, such as the disk peripheral. It provides the link between a file name and a BASIC file number. Once the file has been OPENed.

The general form of the OPEN statement is:

```
OPEN #file number:"file name"[,option[,option[,...]]]
```

In which "option" can be any of the OPEN options available to the user. The user can select the following options:

File organization - SEQUENTIAL or RELATIVE

Open mode - INPUT, OUTPUT, APPEND, or UPDATE

Record type - FIXED or VARIABLE

File life - PERMANENT

4.1.1 File Name Specifications

In order to indicate which drive and which file on it the user wants to access, he must specify a file name in the OPEN statement. This can be in either of two forms:

```
DSKx.file-id or DSK.volname.file-id
```

in which x is the drive ID number (1-3), "volname" is a volume name ID and "file-id" is an individual file ID. Both "volname" and "file-id" can be strings of up to 10 characters long. Legal characters for these strings are all the ASCII characters, except the "." character and the space.

The first form of the file name specification shows the direct drive ID option. The user can specify either DSK1, DSK2, or DSK3 as drive numbers. Only the specified drive is searched for the given file-id.

The second form of the file name specification is the symbolic form. The disk drive is not explicitly assigned, but assigned through the volume name ("volname"). All drives are searched in sequence for the given volname, i.e. DSK1 first, then DSK2, then DSK3. The first drive with the given volname on its disk will be used for the file-id search. It is allowed to use two or more disks with the same volname in the system, however, if the specified file-id doesn't exist on the first drive with the given volname, the other disk drive(s) with the same volname will not be searched.

Whichever form is used, the file-id has to be unique for the selected drive, i.e. if a new file is created, the file-id used must differ from all other file-ids on that drive, or the existing file will be replaced by the new one, unless it is protected.

The file-id in the OPEN statement has to correspond to a data file. If the file was created by a SAVE command, an OPEN for that file will give an error, unless the file is opened for OUTPUT mode, in which case the program file will be replaced by the new data file.

The actual number of Allocatable Units (AUs) allocated can be computed by using the following rules:

1. VARIABLE length records have an overhead of one byte per record plus one byte per AU.

2. Logical records never cross AU boundaries, i.e. an integer number of logical records has to fit in an AU.

A direct result of these rules is that the maximum length of VARIABLE length records is limited to 254 (2 less than the AU size).

Initial allocation of a file is done to avoid scattering of data blocks over a diskette. NOTE: Initial allocation does NOT change the End of File markers, i.e. if 100 records have been initially allocated, the file will still have its EOF set at record 0!!

The initial allocation is only used if a file is opened for OUTPUT mode or if a nonexisting file is opened for UPDATE or APPEND mode. It is ignored if the file is opened for any other case.

4.1.2 File Organization Option

The two file organizations the user can specify are:

1. SEQUENTIAL - Access the file in sequential order, comparable to tape-access. The file may be accessed in any of the four I/O modes. Record type may be specified as FIXED or VARIABLE. File type may be specified as INTERNAL or DISPLAY.

2. RELATIVE - Access the file in random order. The open mode can be any of the available four modes, and the record type must be FIXED, and may be either INTERNAL or DISPLAY. Due to BASIC limitations, the combination RELATIVE and APPEND is not supported. This combination is trapped out as an error.

The default file organization is SEQUENTIAL.

Both the SEQUENTIAL and RELATIVE specifications can optionally be followed by an initial record allocation specification. This spec. indicates the number of records to be allocated initially. In case the record length has been specified as VARIABLE, the allocation will be made for maximum length records.

The number of records initially allocated has to be less than 32767, in order to stay within the record addressing range of a file management system.

4.1.3 Open-mode Operation

BASIC accepts four access modes:

1. INPUT - Data in a file can only be read. The file has to exist before it can be read.
2. OUTPUT - Data can only be written to a file. A new file is created if one does not exist. If one of the same name exists it will be overwritten unless it is protected.
3. APPEND - Data can only be written at the end of the file. If the file does not exist already this mode is equivalent to OUTPUT. Due to the limitations of the console, this mode can only be used for VARIABLE length records.
4. UPDATE - Data can be both written and read. If the file does not exist, it is created. Otherwise data in an existing file can be read and/or changed and new data can be added or old data can be deleted. UPDATE mode is generally used for files OPENed in RELATIVE mode, although SEQUENTIAL is permitted. VARIABLE length record files can be OPENed in UPDATE mode, however, once a new record is written, all the original data behind this record will be lost. This mechanism is mainly intended for use in intermediary files, i.e. first the data is written out, then it is read back without closing the data file.

Note that for UPDATE mode, it is never possible to decrease the size of a file. A re-write will only reset the EOF markers, without releasing the datablocks.

The default OPEN mode is UPDATE, i.e. the file can be both read and written.

```
OPEN #250:"DSK1.FILEA"
```

This statement will open a file called "FILEA" on the disk in drive #1 and its file reference number in BASIC is 250. The attributes assigned to this file are:

File-organization - SEQUENTIAL

Open-mode - UPDATE

Record-type - VARIABLE

File-type - DISPLAY

File-life - PERMANENT

Record length - If none existed before it will be 80 else it will be equal to the length of the file when it was created.

```
OPEN #24:"DSK.MASTER.TABLES",INPUT,RELATIVE,INTERNAL
```

Opens a file called "TABLES" on a disk called "MASTER". Drives will be searched in sequence till one is found with the disk called "MASTER", then it will be searched for a file called "TABLES". If it exists it will be made accessible to BASIC otherwise you get an error. The specifications for this file are:

File-organization - RELATIVE

Open-mode - INPUT

Record-type - FIXED

File-type - INTERNAL

File-life - PERMANENT

Record-length - is equal to the stored length for the file "TABLES".

```
OPEN #1: "DSK3.TESTDATA",OUTPUT,FIXED 40, INTERNAL,RELATIVE
```

Creates a random access file called "TESTDATA" on drive 3. If it exists already, it is overwritten with the new data. The attributes for this file are:

File-organization - RELATIVE

Open-mode - OUTPUT

Record-type - FIXED, 40 characters

File-type - INTERNAL

File-life - PERMANENT

```
OPEN #1:"DSK1.",INTERNAL,FIXED 38,INPUT
```

This command will open the catalog file for sequential input. For more information see section 5.

4.1.4 Record-Type Option

The record-type option is used to specify the size of each record in the file. This size can be either FIXED, all records have the same length, or VARIABLE, with a maximum length optional. If the file organization specified is RELATIVE, the only legal type is FIXED, which is also the default for relative record files.

Both the FIXED and the VARIABLE options can be followed by an expression indicating the actual or maximum record length. Since the length is used to reserve buffer space in the BASIC interpreter, a user is advised to select the length as precisely as possible. Larger record lengths mean fewer variables can be used by BASIC.

The disk peripheral defaults the record lengths for both the FIXED and VARIABLE options to 80 characters.The default record-type for SEQUENTIAL files is VARIABLE; for RELATIVE files it is FIXED.

If a file is opened for any I/O mode other than OUTPUT, and it already exists, the record length, has to match the stored length. If no record length is given the DSR will default to the stored length.

The maximum record length for FIXED records is 255, and for VARIABLE length records it is 254.

4.1.5 File-Type Option

The file-type option can be used to specify the format of data to be stored in the file. There are two formats available:

1. DISPLAY - Stores data in readable format, i.e. as it would be printed on a printer. If the data has to be read back by the machine, this format is not recommended.

2. INTERNAL - Stores data in machine readable format. Since most files on the disk will be read by machine, this format is recommended. It relieves the user of storing separate data like quotes and commas in the file in order to make it suitable for an INPUT command. It avoids the overhead of converting the internal machine representation for the numbers and strings into a representation that is readable for humans and vice versa.

Again, if the file exists and the I/O mode is not OUTPUT, the specification has to match the value stored at file creation. BASIC uses DISPLAY as a default, which means that if data is stored in INTERNAL format, the user always has to indicate this in the OPEN command.

4.1.6 File-Life Option

BASIC only recognized the PERMANENT option as a file-life specification. Since it is also the default it can be eliminated.

4.1.7 Examples

These examples are to clarify the OPEN statement. Remember that whenever an attribute for a file does not match the one stored it will result in an error. However, SEQUENTIAL files can be opened as RELATIVE and vice-versa if the record type was FIXED.

The key word DELETE is optional with the CLOSE statement. In case DELETE is specified, the file is not only disconnected from the file number, but the disk space taken up by the file is released, and the file-id is erased from the disk's catalog. This means the file can no longer be accessed, not even with an OPEN statement (see DELETE statement).

4.2 CLOSE Statement

The CLOSE statement closes the association between the BASIC file-number and the file. After the CLOSE statement is performed, BASIC can no longer access that file, unless it is OPENed again.

The general form of the CLOSE statement is:

```
CLOSE #file-number[:DELETE]
```

A few examples of the CLOSE statements are:

1. `CLOSE #240` - close the file associated with #240

2. `CLOSE #240:DELETE` - same as above, but also deletes the file

4.3 PRINT Statement

The PRINT statement can be used to write information out to a file that has been previously OPENed. The PRINT statement can only be used for files that have been opened for access in either OUTPUT, UPDATE, or APPEND mode. A PRINT to a VARIABLE record length file will always set a new EOF mark, causing data behind the current record to be lost.

The general form of the PRINT statement is:

```
PRINT #file-number[,REC record-number][:print-list]
```

For a detailed description of the PRINT statement, refer to the 99/4 BASIC Language User's Reference Guide.

4.4 INPUT Statement

The INPUT statement can be used to read information from an existing file. The INPUT statement can only be used for files that have been OPENed for access in either INPUT or UPDATE mode.

The general form of an INPUT statement is:

```
INPUT #file-number[,REC record-number]:variable-list
```

For a detailed description of the INPUT statement, refer to the 99/4 BASIC Language User's Reference Guide.

4.5 RESTORE Statement

The RESTORE statement repositions an open file to its first record, or at a specific record if the file is opened for RELATIVE mode and the RESTORE contains a REC clause.

The general form for the RESTORE statement is:

```
RESTORE #file-number[,REC record-number]
```

Generally RESTORE is used to reposition a file for a second read of the same data. However, using the REC clause, the user may position the current access pointer anywhere within or without the file, if the file is opened for RELATIVE mode. In this case the file may be sequentially read, starting at a random point within a file.

If the file is opened for OUTPUT or APPEND mode, the RESTORE statement will not be performed and an error will be given.

4.6 DELETE Statement

The DELETE statement may be used to remove files that are no longer needed from the disk. This will free up space allocated for the file.

The general form for the DELETE statement is:

```
DELETE "file-name"
```

The DELETE statement is a statement for which no previous OPEN is required. Therefore it is possible to DELETE a file which is still OPEN for access. If this happens, any future reference to the file, including a CLOSE, will give an error. An example of the described sequence may be:

```
100 OPEN #2:"DSK1.FILE",OUTPUT
110 PRINT #2:"HELLO"
120 DELETE "DSK1.FILE"
130 CLOSE #2
```

Here line 130 will give an error, since the file no longer exists at that point in the program.

4.7 OLD Command

The OLD command allows for retrieval of previously stored programs from a disk. The program must have been stored with a SAVE command, since the disk software will not allow loading of a data file with the OLD command.

The general form for the OLD command is:

```
OLD file-name
```

Since OLD is a system command that cannot be used in a program, the file-name can be an unquoted string.

```
OLD DSK1.PROGRAM
```

Is perfectly legal.

4.8 SAVE Command

The SAVE command can be used to save the current program in the 99/4 onto a disk file, which then can be reloaded with an OLD command.

The general form for the SAVE command is:

```
SAVE file-name
```

Like OLD, SAVE is a system command, allowing the user to type the file name without quotes. SAVE will create a new file overwriting any existing file with the same name unless it has been protected.

4.9 EOF Function

The EOF function can be used to test for the end of file during I/O operations. Three conditions are indicated by the EOF routine:

0 Not EOF (End of File)

1 Logical EOF (End of File)

-1 Physical EOM (End of Medium)

Physical EOM can only be detected if the device is at its physical end and the file is at its logical end.

The general form for the EOF function is:

```
EOF(file-number)
```

The EOF indication only has meaning in the case of sequential access to files, since for random access the next record to be read or written cannot be determined from the current one. Therefore, the EOF subroutine will assume that the next record to be read/written is the sequentially next record.

The logical EOF indicates that the next sequential read/write operation will attempt to access a record outside the current file. In general this indication will only be used for read operations.

Because of pending BASIC INPUT conditions, it is possible that the EOF subroutine indicates "EOF", even if the next INPUT statement will yield no EOF error, since it can read data from the current record. Something similar can happen if it indicates "no EOF" and the next INPUT statement reads more than one record. In this case the INPUT might be terminated with an error. To avoid this type of situation, the user is advised to use only non-pending INPUT statements (INPUT statements without a trailing comma), so that each record corresponds to one INPUT statement.

For random access to files, the EOF subroutine can only give meaningful results if the access is converted to "semi-sequential" access, i.e. if the record is positioned through a RESTORE statement and then sequentially accessed through any I/O statement without REC clause specification. After the RESTORE the EOF subroutine will indicate that the condition for the next record is (EOF, EOM or available), without issuing an I/O error.

Note that there is one EOM condition that cannot be detected by the EOF subroutine. This condition occurs when the datablocks on a disk become so scattered that not enough datablocks can be allocated for a file. In this case a PRINT operation will be aborted with an I/O error, even though there is enough space available on the disk, and the EOF function does not indicate an EOM condition.

Note however that the software file protection does not offer any protection against complete disk re-initialization. The only way to avoid file loss in that case is to "write protect" the disk itself by placing a tab over the notch on the right side of the disk. This will disallow any write operation to the disk, giving a hard error as soon as the disk is being accessed for write operations. Notice that this type of write protection is only intercepted on the actual write operation. The disk software will not disallow destructive access to the disk until the moment it actually tries to modify part of the disk.

Chapter 5

CATALOG FILE ACCESS FROM BASIC

The BASIC user can access a disk catalog like a read-only disk file. This disk file has no name and is of the INTERNAL, FIXED length type. An example of a CATALOG file OPEN is:

```
OPEN #1:"DSK1.",INPUT,INTERNAL,RELATIVE
```

Since BASIC will automatically default the record length to the current value, it is recommended that the user not specify this length. If he wishes it is 38. Every other record length will result in an error.

The CATALOG file acts like it is protected, as it will only allow INPUT access. An attempt to open the CATALOG file for any other mode will result in an error.

The data in the CATALOG file is written in the standard BASIC INTERNAL format. Every record in the file contains four items: one string and three numerics. The string indicates the name of the disk, containing up to 10 characters. The numeric items indicate the following:

1. Record-type - Always zero for this record.
2. Total number of AUs on the disk - for a standard 40-track disk this should be 358.
3. Total number of free AUs on the disk.

Record numbers 1-127 contain information about the corresponding file in the CATALOG. Non-existing files will give a null-string as the first item, and 0s for the remaining three items. Existing files will indicate the file name in the string item, and the following in the numeric items:

- File-type - negative if file is protected.
    1. DISPLAY/FIXED datafile
    2. DISPLAY/VARIABLE datafile
    3. INTERNAL/FIXED datafile
    4. INTERNAL/VARIABLE datafile
    5. Memory image file (e.g. BASIC program)
- Number of AUs allocated by the file.

- Number of bytes per record.

A type 5 file will always indicate a 0 in its third item, since the number of bytes per record has no meaning.

Chapter 6

FILE PROTECTION

A user may select to protect any of the files on a disk. This can be done with the disk manager package.

The effect of the protected file is that the system disallows any type of destructive access to that file, the following actions are disabled.

- SAVE to a protected file.
- OPEN a protected file in a mode other than INPUT.

Chapter 7

FILES SUBPROGRAM

The default number of files that can be opened simultaneously is three. To modify this number, the FILES subprogram has been provided. The syntax for this subprogram is:

```
CALL FILES(x)
NEW
```

Where "x" is a number from 1-9, indicating the number of files that can be opened at once. Arithmetic expressions and variables are not allowed.

The NEW command following the FILES call has to be considered as part of the FILES call, since FILES will destroy some pointers used by the BASIC interpreter.

WARNING

The usage of the FILES subprogram in a BASIC program is not allowed, and doing so will cause strange results. Likewise a call to FILES without a NEW command immediately following it may cause strange results, ranging from loss of program to loss of data on diskettes. The only way to avoid this is to use the FILES subprogram only in the above defined manner!

The FILES subprogram will check only for the above defined syntax.

CALL FILES(2)*2 will execute the same as CALL FILES(2)

The disk has a standard overhead buffer of 534 bytes. Each open file adds 518 bytes to this buffer area for the disk. If this would leave the user with a buffer of less than 2K bytes as may occur in a 4K system, the files program will return with an INCORRECT STATEMENT error.

In case of a syntax error before the right parenthesis (")"), an INCORRECT STATEMENT will occur.

Chapter 8

I/O ERROR CODES

I/O errors detected by the disk peripheral software are always indicated by BASIC in the following format:

```
* I/O ERROR xy [IN 111]
```

The digits "xy" indicate the type of error that has occurred. The first digit (x) indicates the I/O routine in which the error occurred. The following I/O routine codes can be given:

0 Error in OPEN routine

1 Error in CLOSE routine

2 Error in READ routine

3 Error in WRITE routine

4 Error in RESTORE routine

5 Error in LOAD routine used during OLD

6 Error in SAVE routine

7 Error in DELETE routine

9 Error in STATUS routine used in EOF

The second digit (y) indicates the type of I/O error that has occurred. There are 8 different codes with the following meaning:

0 BAD DEVICE NAME - the device could not be found

1 DEVICE WRITE PROTECTED - unprotect the disk and try again

2 BAD OPEN ATTRIBUTE - one or more open options were illegal and didn't match the file characteristics.

3 ILLEGAL OPERATION - should not be generated by BASIC for the disk peripheral. Indicates usage of non-existing I/O code.

4 OUT OF SPACE - a physical end of the file was reached, there was insufficient space on the disk to complete the operation.

5 ATTEMPT TO READ PAST EOF

6 DEVICE ERROR - a hard or soft device error was detected. This may occur if the disk was not initialized or was damaged, the system was powered down during disk writes, the unit did not respond, etc.

7 FILE ERROR - the indicated file or volume doesn't exist; the file type doesn't match the access code (program file versus data file).