Please do not upload this copyright pdf document to any other website. Breach of copyright may result in a criminal conviction.

This Acrobat document was generated by me, Colin Hinson, from a document held by me. I requested permission to publish this from Texas Instruments (twice) but received no reply. It is presented here (for free) and this pdf version of the document is my copyright in much the same way as a photograph would be. If you believe the document to be under other copyright, please contact me.

The document should have been downloaded from my website https://blunham.com/Radar, or any mirror site named on that site. If you downloaded it from elsewhere, please let me know (particularly if you were charged for it). You can contact me via my Genuki email page: https://www.genuki.org.uk/big/eng/YKS/various?recipient=colin

You may not copy the file for onward transmission of the data nor attempt to make monetary gain by the use of these files. If you want someone else to have a copy of the file, point them at the website. (https://blunham.com/Radar). **Please do not point them at the file itself as it may move or the site may be updated.**

It should be noted that most of the pages are identifiable as having been processed by me.

I put a lot of time into producing these files which is why you are met with this page when you open the file.

In order to generate this file, I need to scan the pages, split the double pages and remove any edge marks such as punch holes, clean up the pages, set the relevant pages to be all the same size and alignment. I then run Omnipage (OCR) to generate the searchable text and then generate the pdf file.

Hopefully after all that, I end up with a presentable file. If you find missing pages, pages in the wrong order, anything else wrong with the file or simply want to make a comment, please drop me a line (see above).

It is my hope that you find the file of use to you personally – I know that I would have liked to have found some of these files years ago – they would have saved me a lot of time !

Colin Hinson
In the village of Blunham, Bedfordshire.

FUNCTIONAL SPECIFICATION

FOR THE

99/4 DISK PERIPHERAL

Consumer Group
Mail Station 5890
2301 N. University
Lubbock, Texas 79414

TEXAS INSTRUMENTS
INCORPORATED

Date: March 28, 1983
Version 3.0

TABLE of CONTENTS

Paragraph Title

SECTION 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 a BASIC user standpoint.

This specification will not describe the utility package which is build into the disk controller, nor will it describe the GPL interface routines for direct disk access. These special topics are discussed in a separate document, the GPL Interface Specification for the 99/4 Disk Peripheral.

SECTION 2

APPLICABLE DOCUMENTS

File Management Specification for the TI-99/4 Home Computer
(Version 2.5, 25 February 1983)

Home Computer BASIC Language Specification
(Revision 4.1, 12 April 1979)

Home Computer Disk Peripheral Hardware Specification

Software Specification for the 99/4 Disk Peripheral
(Version 2.0, Revised 28 March 1983)

GPL Interface Specification for the 99/4 Disk Peripheral
(Version 2.0, Revised 28 March 1983)

SECTION 3

SUPPORTED FILE MANAGEMENT OPTIONS

The disk peripheral supports most of the options decribed in the File Management Specification for the TI-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 indicate the name of the file that is to be opened, and the drive identification or the diskette name (assigned at diskette initialization).

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 that can be open simultaneously is limited, it is advised that each file is closed as soon as it is no longer needed.

READ - Read a logical record from the opened file.

WRITE - Write a logical record to the opened 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 for 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, whereby the drive can be specified either by the drive name or the disk name. DELETE frees up the space occupied by the file for future use.

SCRATCH RECORD - This function is not supported by the disk peripheral.

STATUS - Indicate current status of a file. This includes the logical and physical EOF flags and the protection flag.

SECTION 4

INTERFACE TO BASIC

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

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 user can access it through the PRINT and INPUT statements, depending upon the mode for which 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-type - INTERNAL or DISPLAY
File-life - PERMANENT

4.1.1 File-name Specification.

In order to indicate which disk drive and which file on that disk drive the user wants to access, he has to specify a file-name in the OPEN statement. For the disk peripheral this file-name can be either of two forms:

DSKx.file-id

or

DSK.volname.file-id

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

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

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

The file-id indicated in the OPEN statement has to correspond to a data file. If the file indicated 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.

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, the record-type must be FIXED (which is also the default value for this file-organization), and the file-type

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 the RELATIVE specification can optionally be followed by an initial record allocation specification. This specification 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 the file management system.

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

1. VARIABLE length records have an overhead of 1 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 (two 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 non-existing file is opened for UPDATE or APPEND mode. It is ignored if a file is opened for any other case.

4.1.3 Open-mode Option.

BASIC accepts four access modes:

1. INPUT - data in the file can only be read. The file has to exist before it can be read.

2. OUTPUT - data can only be written to the file. A new file is created if the file doesn't already exist. If a file of the same name already exists, the original data in that file will be lost, unless the file is protected.

3. APPEND - data can only be written at the end of the file. If a file of the given name doesn't already exist, this mode is equivalent to OUTPUT. Due to limitations in the console, this mode can only be used for VARIABLE length records.

4. UPDATE - data can both be 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 access 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 End of File markers, without releasing the datablocks.

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

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, i.e. all records have the same length, or VARIABLE, with a given (maximum) length optional. If the file-organization specified is RELATIVE, the only legal record-type specification is FIXED, which is also the default for relative record files.

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

2. OUTPUT - data can only be written to the file. A new file is created if the file doesn't already exist. If a file of the same name already exists, the original data in that file will be lost, unless the file is protected.

3. APPEND - data can only be written at the end of the file. If a file of the given name doesn't already exist, this mode is equivalent to OUTPUT. Due to limitations in the console, this mode can only be used for VARIABLE length records.

4. UPDATE - data can both be 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 access 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 End of File markers, without releasing the datablocks.

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

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, i.e. all records have the same length, or VARIABLE, with a given (maximum) length optional. If the file-organization specified is RELATIVE, the only legal record-type specification is FIXED, which is also the default for relative record files.

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

The disk peripheral defaults the recordlength for both th FIXED and the VARIABLE option to 80 characters. The defau. 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 the file already exists, the record length, if given, has to match the previously stored length exactly. If no record length is given, the disk DSR will automatically default to the stored length.

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

4.1.5 File-type Option.

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

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

2. INTERNAL - store the data in a machine readable format. Since most of the datafiles on the disk will be read by the machine, this data format is recommended. It relieves the user of the burden of storing separation data (like quotes and commas) in the file in order to make it suitable for an INPUT command. It also avoids the overhead of converting the internal machine representation for 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 given specification has to match the value stored at file creation. BASIC will use 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 recognizes the PERMANENT option as a file-life specification. Since this is also the default, this specification can be omitted completely.

4.1.7 Examples.

The following examples are meant to clarify the usage of the OPEN statement. Please remember that whenever the given attributes for a file don't match the attributes stored when the file was created, an error will be given. However, SEQUENTIAL files can be opened for RELATIVE access, and vice versa, if the record-type specified was FIXED.

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

This statement will open a file called "FILEA" on disk drive #1 for access as BASIC file number 250. The specific attributes assigned to this file are:

```
File-organization - SEQUENTIAL
Open-mode - UPDATE
Record-type - VARIABLE
File-type - DISPLAY
File-life - PERMANENT
```

The record length depends upon the existance of the file. If the file exists, the record length will be equal to the length used when the file was created. If the file doesn't exist yet, the record length will be 80 characters.

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

Open a file called "TABLES" on a disk called "MASTER". The disk drives will be searched in sequence, and the first disk found called "MASTER" will be searched for a file called "TABLES". If that file exists, it will be made accessable for BASIC as file number 24. If it doesn't exist, an error will be indicated. The specific attributes assigned to this file are:

```
File-organization - RELATIVE
Open-mode - INPUT
Record-type - FIXED
File-type - INTERNAL
File-life - PERMANENT
```

The record length is equal to the stored length for the file "TABLES".

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

Create a random access file called "TESTDATA" on drive #3. If the file already exists, overwrite it with the new data (the file-name has to be unique). The attributes created 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 refer to section 5.

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 specific file, unless it is OPENed again.

The general form of the CLOSE statement is:

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

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

A few examples of CLOSE statements are:

```
CLOSE #240          Close the file associated with #240.

CLOSE #240:DELETE   Same as above, but also delete 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 End of File 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, the user is referred to the 99/4 BASIC Language User's Reference Guide.

4.4 INPUT Statement

The INPUT statement can be used to read information from a previously created and OPENed 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 the INPUT statement is:

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

A more detailed description of the INPUT statement can be found in 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 of 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 a file may also be sequentially read, starting at a random point within the 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 nc longer needed from a disk. This will free up the space allocated for the file.

The general form of 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 that file, including a CLOSE, will give an error indication. 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
```

In this case line 130 will give an error, since the file "DSK1.FILE" will no longer exist at that point in the program.

4.7 OLD Command

The OLD command allows for retrieval of previously store programs from a peripheral like a disk. The program must have been stored with the SAVE command, since the disk softwarewill not allow for the loading of data files with the OLD command.

The general form of 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, i.e. the command

```
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 can then be reloaded with the OLD command.

The general form of 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 automatically create a new file, overwriting any existing file of the same name, unless this file has been protected.

4.9 EOF Function

The EOF function can be used to test for end of file during I/O operations. Three file 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 of 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, since for write operations a logical EOF will be indicated as soon as records are appended at the end of the existing file.

SECTION 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 correct value, it is recommended that the user does not specify this length. If, for whatever reason, the user does want to specify this length, it has to be specified as 38. Every other record length will result in an error.

The CATALOG file acts like it is protected, i.e. 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. There are exactly 128 records in the CATALOG file, numbered from 0 to 127.

Record number 0 contains information about the volume on which the CATALOG file is located. The string indicates the name of the disk, containing up to 10 characters. The numerical items indicate the following:

1. Record-type - always 0 for this record.
2. Total number of AUs on the disk - for a standard 40-track diskette this should be 358.
3. Total number of free AUs on the disk.

Record numbers 1 through 127 contain information about the corresponding file in the CATALOG. Non-existing files will give a null-string as 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 (memory image) will always indicate a zero in its third item, since the number of bytes per record has no meaning.

SECTION 6

FILE PROTECTION

A user may select to protect or unprotect any of the files on a disk. This can be done with the Disk Manager Package.

The effect of a protected file is that the system automatically disallows any type of (potentially) destructive access to that specific file, i.e. the following actions are disabled:

SAVE to a protected file.

OPEN a protected file for an access mode other than INPUT.

Note however that software file protection does not offer any protection against complete disk re-initialization. The only way to avoid file loss in that specific case is to "write protect" the disk itself by placing a write protect 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 kind of write protection is only intercepted on the actual write operations, i.e. the disk software will not disallow potentially destructive access to the disk up to the moment that it actually tries to modify part of the disk.

SECTION 7

FILES SUBPROGRAM

The default number of files that can be open simultaneously is 3. 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 to 9, indicating the number of files that can be opened simultaneously. Arithmetic expressions and variable names are not allowed in the FILES subprogram.

The NEW command following the FILES call has to be considered a part of the FILES call, since FILES will destroy some pointers used by the BASIC interpreter. The user is urged to issue a NEW command after each call to the FILES subprogram.

WARNING

> The usage of the FILES subprogram in a BASIC program is not allowed, and doing so will cause unpredictable and usually highly undesirable results. Likewise a call to FILES without a NEW command immediately following it may cause unpredictable results, ranging from loss of program to loss of data in 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. Any characters following the call are ignored, i.e. the call

```
CALL FILES(2)*2
```

is perfectly legal, and will be executed the same as

```
CALL FILES(2)
```

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

In case a syntax error is detected before the right parenthesis (")"), an INCORRECT STATEMENT error will be indicated.

SECTION 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 specified 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 or 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 file was reached, i.e. there was insufficient space on the disk to complete the requested 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 given unit didn't respond, etc.

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

GPL INTERFACE SPECIFICATION

FOR THE

99/4 DISK PERIPHERAL

Consumer Group
Mail Station 5890
2301 N. University
Lubbock, Texas 79414

TEXAS INSTRUMENTS
INCORPORATED

TABLE of CONTENTS

Paragraph Title

LIST of TABLES

SECTION 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 either a preceding 0 or a preceding >. Therefore the numbers 010 and >10 are the same as 16 decimal.

The items in transfer blocks which are enclosed in brackets {} are items that are returned by the subprogram.

SECTION 2

APPLICABLE DOCUMENTS

File Management Specification for the TI-99/4 Home Computer
(Version 2.5, 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, 28 March 1983)

Software Specification for the 99/4 Disk Peripheral
(Version 2.0, Revised 28 March 1983)

SECTION 3

DISK DSR LEVEL CONCEPT

The disk DSR has been developed as a three level software package, each level defining 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 functions like sector read/write, head control, drive selection, and track formatting.

Level 2 - Definition of the "file" concept. Each file is addressable by its 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 management 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 routines in the disk 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 a double density disk would only involve changing the routines on this level, as long as the physical sector size remains 256 bytes.

The following routines are available on this level:

Sector read/write

Format Disk

>8356 must point to name length byte in VRAM
Name length for subprogs is >01, name is as given below. ie/ read/write - >10

.e following sections will contain a description of these)utines and their call requirements. All parameters will be ansferred through the FAC block in CPU RAM. This block is ocated in CPU RAM starting at relative location 04A (currently 834A).

.1.1 Sector READ/WRITE - SUBPROGRAM >10.

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

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 single density mini-floppy) rather than as a track and sector number, which would require a 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 on which the operation is to be performed. This entry has to be either a 1, 2, or 3.

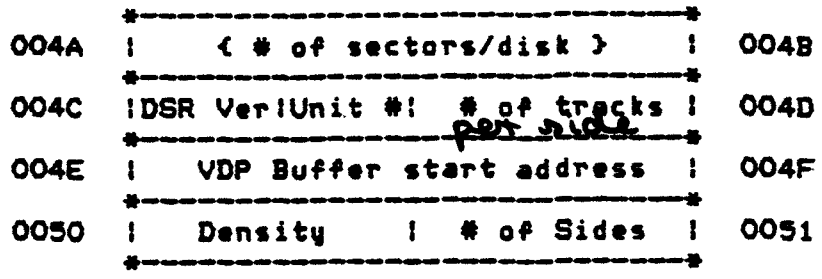
READ/WRITE - Indicates the direction of data-flow.
0 = WRITE
0 = READ

VDP Buffer start address - Indicates start of VDP 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 Formatting - SUBPROGRAM 011.

The transfer block for this subprogram is:



The meaning of each entry is:

# of sectors/disk - Is returned by the routine to provide compatibility 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 on any version of the DSR.
1 indicates the format requires the 2nd version of the DSR for 1 of 2 reasons. It may be because 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 available on the 1st or 2nd DSR. (Density and perhaps double tracking if it is available on the next DSR.)

Unit # - Indicates the disk drive on which the operation is to be performed. This entry has to be either a 1, 2, or 3.
This is the LSNibble.

# of tracks - Indicates the number of tracks to be formatted. In the current version this entry has to be either 35 or 40!!! Upon return, this entry contains the number of sectors/track.

VDP Buffer start address - Indicates start address of the VDP buffer that can be used by the disk controller to write tracks. Must contain complete track data, inc gaps, side, sector No etc. Total size = L.T. >D00 bytes

Density -

# of Sides - Indicates the number of sides to format.

This routine will format the entire disk on the 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 the 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 bytes.

Error codes are returned in CPU location 050.

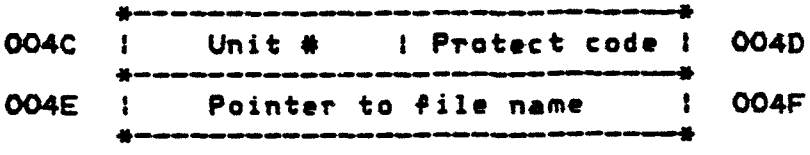
3.2 Level 2 Subroutines

The Level 2 Subroutines are those routines that use the concept "file" rather than "logical sector 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 logical blocks on this level are accessed by filename and logical block offset. This offset starts with block 0 and ends with block N-1 for a file with a length of N blocks.

3.2.1 Modify File Protection - SUBPROGRAM 012.

The transfer block for this subprogram is:



The protect 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 protect 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 a file-name. The first pointer points to the new name, the second one to the original filename. 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 filename. 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 04E.

Error codes are returned in the standard error byte at CPU location 050. The error codes returned are identical to the standard file management error codes, i.e. only the upper thr 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 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 very 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 File - 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. This entry has to be either a 1, 2, or 3.

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

0 Transfer file parameters. This will transfer Level 2 parameters to the additional information area (six 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 from the 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 a 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 additional 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 allocated	

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

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

The remaining 6 bytes are explained in the Software Specification for the 99/4 Disk Peripheral. The user should be very careful when changing these bytes, since they directly affect Level 3 operation. If the information in these 6 bytes is not modified consistently, unpredictable results may occur.

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 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. This entry has to be either a 1, 2, or 3.

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

0 Create file and copy Level 3 parameters from additional information area.

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

Pointer to file name - Contains a pointer to the first character of a 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 additional information 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 allocated	

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

The # of first AU entry indicates the AU number at which the read should begin. If the access code = 0 (parameter

passing), the total number of AUs to be allocated for the file has to be indicated.

The remaining 6 bytes are explained in the Software Specification for the 99/4 Disk Peripheral. The user should be very careful when changing these bytes, since they directly affect Level 3 operation. If the information in these 6 bytes is not modified consistently, unpredictable results may occur.

Error codes are returned at location 050 in CPU RAM.

3.4 Buffer Allocation Routine - SUBPROGRAM 016

The argument for this subprogram is the number of file buffers to be allocated. 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 disk usage to facilitate the simultaneous opening of the given number of files. This number has to be between 1 and 16.

The disk software automatically relocates all buffer area 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 CRU address for 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 screen and data storage. If this is not the case, or if the total number of buffers requested is 0 or >16, the request is ignored and an error code will be indicated in CPU location 050 (currently >8350).

Successful completion is indicated by a 0 byte in CPU location 050. A nonzero byte in CPU location 050 indicates unsuccessful completion.