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

SECTION 2   APPLICABLE DOCUMENTS

SECTION 3   SUPPORTED FILE MANAGEMENT OPTIONS

SECTION 4   INTERFACE TO BASIC

SECTION 5   CATALOG FILE ACCESS FROM BASIC

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 <u>File Management Specification for the TI-99/4 Home Computer</u>. 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.

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 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
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 no 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 stored 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.

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 EOF 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, i.e. INPUT statements without a trailing comma, and to know what type of data has been stored in the records, 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 pointer 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 what 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 data blocks on a disk become so scattered that not enough data blocks 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.

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).