

# Disk MicroColor Basic 2.4

For Alice / MC-10

## DISK OPERATIONS

This version of BASIC provides disk-based file operations on 720K (double density) 3.5 inch floppy disks. Disks must be formatted for use with the FAT12 file system and an allocation cluster size of 4K (eight 512 byte sectors per cluster). A DSKINI command is provided to format disks in this fashion. Only files in the root directory may be accessed from BASIC (a maximum of 112 files per disk). Long file names are not supported.

In BASIC, file names are implemented as string values and have the following format:

```
drive:name.extension
```

The drive number is optional and may be either 0 or 1. If you do not specify a drive number in a file name string then the **default drive** will be used (set with the DRIVE command). The name is required and can be from 1 to 8 characters in length. The extension is optional and can be from 1 to 3 characters in length. Here are some examples of legal file names:

```
PROGRAM.BAS  
1:DATAFILE  
0:NAMES.LST
```

You cannot use any of the following characters in the name or extension:

```
/ * . " : < = > ? + , ; = [ ]
```

## **BACKUP** *source TO destination*

This command copies the entire contents of the disk in the *source* drive to the disk in the *destination* drive. You must have two floppy drives to use this command (*source* and *destination* cannot be the same).

### **WARNING:**

*The BACKUP command will erase any program that is currently in memory.*

```
BACKUP 0 TO 1  
BACKUP 1 TO 0
```

## **CHAIN "filename", line number**

Loads and executes a BASIC program file from disk without erasing any variables that are currently in memory. If you do not specify an extension in the *filename* then the default extension **BAS** is used. The optional line number specifies the first line in the program to execute.

```
CHAIN "LINKER"  
CHAIN "PROGRAM2",100
```

## **CLOSE #buffer**

Closes the data file associated with the buffer, or closes ALL files if no buffer is specified. See **OPEN** for more information.

```
CLOSE #1  
CLOSE
```

## **COPY "file1" TO "file2"**

Copies the contents of *file1* to the new or existing *file2*. You must include the extension in both file names (no default extensions are applied).

```
COPY "PROG1.BAS" TO "PROG2.BAS"  
COPY "0:NAMES.DAT" TO "1:NAMES.DAT"
```

## **DIR drive**

### **DIRI drive**

Displays the disk directory for the specified *drive*. If you do not provide a drive number then the default drive is used (see the **DRIVE** command). The **DIR** command shows only files that are not hidden. The **DIRI** command shows all files, including hidden files.

```
DIR  
DIRI 1
```

## **DOS drive**

Loads and executes a boot module from sector 0 of the disk in the specified *drive*. The drive number is optional. If you do not specify a drive then the default drive is used.

```
DOS  
DOS 1
```

## **DRIVE** *drive*

Sets the default drive number to be used when no drive number is specified in a filename string. The *drive* number must be either 0 or 1. When the computer is powered on, the default drive number is set to 0.

```
DRIVE 1
```

## **DSKINI** *drive, Q, V, I skip\_factor, "label"*

Formats or erases the floppy disk in the specified *drive*. The drive number is NOT optional.

If the **Q** option is included then a **Quick Erase** is performed. This will only format track 0 which contains the directory.

If the **V** option is used then a **Verify** pass of the entire disk is performed. Any bad sectors discovered during the verification will be mapped out of the file allocation table. If more than 8 bad sectors are found then the operation is aborted and an ?IO ERROR will be produced.

The **I** option is optional and, if used, must be followed by a number from 0 to 7. It controls the interleaving of sectors on each track. The default value is 1. You should generally not use a skip factor other than the default value.

The *label* is optional. It can be a string from 1 to 11 characters. If you include a label, it will be displayed whenever you list the disk's directory (DIR).

```
DSKINI 0
DSKINI 0,V
DSKINI 0,I3
DSKINI 1,Q,"GAMES DISK"
```

## **EOF** (*buffer*)

This function is used to determine if the End-Of-File has been reached when reading from a data file. The *buffer* is the number that was used to open the file. If the end of file has been reached then the function returns -1, otherwise it returns 0.

```
IF EOF(1)<>0 THEN CLOSE #1:GOTO 250
```

## **FREE** (*drive*)

The FREE function returns the number of bytes available on the disk in the specified *drive*.

```
PRINT "FREE SPACE ON DRIVE 0 ="; FREE(0)/1024; "K"
```

## **INPUT #*buffer*, *variable\_list***

Reads data from the file associated with *buffer* and assigns each data item from the file to the variable names you specify.

```
INPUT #1, N$, I, J
```

## **KILL "*filename*"**

Deletes the specified file. You must include the extension in the file name when using this command (no default extension is applied).

```
KILL "PROGRAM.BAS"
```

## **LOAD "*filename*"**

Loads a BASIC program file into memory from disk. If you do not specify an extension in the filename then the default extension **BAS** is used.

```
LOAD "CROCKY"  
LOAD "1:CASINO.PGM"
```

## **LOADM "*filename*", *offset***

Loads a machine-language file from disk. If you do not specify an extension in the filename then the default extension **BIN** is used. The *offset* is optional. If provided, the offset is added to the normal load and execution addresses.

```
LOADM "PINBALL"  
LOADM "MLCODE.OBJ", 2000
```

## **LOAD\* *array*, "*filename*"**

Loads data from disk into the specified numeric *array*. If you do not specify an extension in the filename then the default extension **ARY** is used.

The array must have been previously allocated with a DIM statement and be sufficiently large to hold the data contained in the file, otherwise an ?OM ERROR will be produced.

```
LOAD* Z, "NUMBERS"
```

## **MERGE "filename"**

Loads a BASIC program file from disk and merges it with the current program already in memory. If you do not specify an extension in the filename then the default extension **BAS** is used. The merge operation may take a long time to complete when working with large programs.

```
MERGE "OVERLAY.BAS"
```

## **OPEN "mode", #buffer, "filename"**

Opens a file buffer in memory for transferring data to or from a file. The *mode* specifies the type of file access desired:

```
I   Input
O   Output
A   Append
```

The INPUT# and PRINT# statements are used to read or write data to the file. The CLOSE statement should be used when access to the file is no longer needed.

The *buffer* is a number from 1 to 4. Up to four files may be opened at the same time.

If you do not specify an extension in the filename then a default extension of DAT is used.

```
OPEN "I", #1, "GAMEDATA"
```

**WARNING:** *You must CLOSE a file that has been opened using the Output or Append modes before removing the disk from the drive. Failure to do so can result in a corrupted disk.*

## **PRINT #buffer, data\_list**

Prints data to the file associated with *buffer*. You can use a comma (,) or a semicolon (;) to separate each item in the data list.

```
PRINT #1, "JOHN", "SALLY", "FRED"
PRINT #1, N$; X; Y; Z
```

## **RENAME "oldname" TO "newname"**

Changes the name of an existing file. You must include the appropriate extensions in both file names (no default extensions are applied).

```
RENAME "DATAFILE.DAT" TO "DATAFILE.BAK"
```

### **RUN "filename", line\_number**

Loads and executes a BASIC program file from disk. If you do not specify an extension in the filename then the default extension **BAS** is used. The optional *line number* specifies the first line in the program to execute.

```
RUN "CROCKY"  
RUN "1:EDITOR.PGM", 100
```

### **RUNM "filename"**

Loads and executes a Machine Language program file from disk. If you do not specify an extension in the filename then the default extension **BIN** is used. This command is the equivalent of using **LOADM** followed by **EXEC**.

```
RUNM "PINBALL"
```

### **SAVE "filename", A**

Saves the current BASIC program to disk. If you do not specify an extension in the filename then the default extension **BAS** is used. The **A** option saves the program as ASCII text instead of in the normal compressed format.

```
SAVE "PROGRAM"  
SAVE "WUMPUS.TXT", A
```

### **SAVEM "filename", start, end, exec**

Saves the contents of memory from the address specified by *start* to the address specified by *end*. If you do not specify an extension in the filename then the default extension **BIN** will be used.

The optional *exec* argument is the execution address for any machine language code saved in the file (the default address for **EXEC** following **LOADM**). If you do not specify an address for *exec* then the *start* address will be used instead.

```
SAVEM "ML FILE", &H5000, &H53FF, &H5004
```

### **SAVE\* array, "filename"**

Saves the contents of a numeric *array* to disk. If you do not specify an extension in the filename then the default extension **ARY** will be used.

```
SAVE* Z, "NUMBERS"
```

**SET FILE "*filename*", A, I, P**

**RESET FILE "*filename*", A, I, P**

These commands can be used to set or reset individual attributes for a file. The `SET FILE` command assigns attributes to the file, whereas the `RESET FILE` command removes them.

The **A**, **I** and **P** options identify the specific attribute(s) to be set or reset:

A	Archive	Used by Backup programs to identify modified files.
I	Invisible	Not displayed when using the standard <code>DIR</code> command.
P	Protected	Prevents modification or deletion of the file.

```
SET FILE "GAMEDATA.DAT" , I , P
RESET FILE "PROGRAM.BAS" , A
```

## Additional Commands and Functions

### **CLEAR ERROR**

This statement can be used to reset the values returned by the **ERRN** and **ERRL** functions to -1 and 65535 respectively. These are the same values which those functions return when no error has occurred since the program was `RUN`.

### **CSAVEM "*filename*", *start*, *end*, *exec***

Saves a block of memory to cassette. The *start* and *end* arguments specify the inclusive address range of the memory block to save. The *exec* argument is optional and specifies the default address to use for the `EXEC` command when the file is later loaded using `CLOADM`. If *exec* is omitted then *start* will be used instead.

### **DEFUSR = *address***

Defines the address for the machine language `USR` function.

```
DEFUSR = &H5000
```

## **DEL start-end**

Deletes one or more program lines. The lines to be deleted are specified as a range of line numbers in the same format accepted by LIST.

DEL 50	delete line 50
DEL -30	delete all lines before and including line 30
DEL 100-200	delete all lines from 100 to 200 inclusive
DEL 500-	delete line 500 and all following lines

## **E line** (Alice 4K / MC-10 only)

Enters Edit mode for the specified *line* number. The following key commands are available in Edit mode:

A	Abort any changes made to the line and start over.
nB	Move cursor back <i>n</i> characters.
nC	Change <i>n</i> number of characters. (type the new characters after C)
nD	Delete <i>n</i> number of characters.
I	Begin <i>Insert</i> mode.
H	Delete remainder of the line and begin <i>Insert</i> mode. (HACK)
nKc	Delete characters up to the <i>n</i> th occurrence of character <i>c</i> . (KILL)
L	List the line (including changes made so far) and continue editing.
nN	Move cursor ahead <i>n</i> characters. The spacebar may also be used for this.
nSc	Move ahead to the <i>n</i> th occurrence of character <i>c</i> . (SEARCH)
X	Move to the end of the line and begin <i>Insert</i> mode. (EXTEND)
Cntrl-Z	Exit from <i>Insert</i> mode.
BREAK	Cancel the edit operation without saving changes.

**NOTE:** *The Editor is automatically invoked when a Syntax Error occurs in a program line.*

## **ELSE**

You may include an ELSE clause in an IF statement. A line number immediately following ELSE is an implied GOTO.

```
IF I >= 25 THEN PRINT I ELSE PRINT I*2
IF SGN(X)= -1 THEN 100 ELSE 200
```

## **ERRL**

Returns the program line number in which the last error occurred. If no error has occurred since the program started running or since the last CLEAR ERROR statement was executed then 65535 will be returned.

## ERRN

Returns a number representing the type of error which last occurred. See **ERROR CODES** on page 13 for a list of possible values.

If no error has occurred since the program started running or since the last `CLEAR ERROR` statement was executed then -1 will be returned.

## ERROR *number*

Simulates an error of the type specified by the error *number*. See **ERROR CODES** on page 13 for the list of error numbers.

## HEX\$ (*number*)

The HEX\$ function returns a string representing the hexadecimal value of *number*. The argument must be within the range of 0 to 65535.

## INSTR (*position, subject, target, instance*)

The INSTR function searches the *subject* string for an instance of the *target* string. The value returned is either the position where the instance was found or 0.

The *position* argument is optional and specifies the first character position within the *subject* string where searching will begin. If *position* is omitted then searching begins from the first character in the *subject* string. If *position* is greater than the number of characters in *subject* then the function returns 0.

The *instance* argument is also optional and allows you to search for a specific instance of *target* in situations where *target* may occur more than once within the *subject* string. If *instance* is omitted then INSTR will search for the first instance of *target*. You can search for the last (right-most) instance of *target* by supplying an *instance* value of 0.

## JOYSTK (*player*)

Read the current state of the left or right joystick. Pass 0 for *player* to read the left joystick or 1 to read the right joystick. The value returned is a 5 bit integer in which each bit corresponds to one of the joystick directions or the "fire" button:

Bit	Value	Condition
0	1	Up
1	2	Down
2	4	Left
3	8	Right
4	16	Fire

```
IF JOYSTK(0) AND 1 THEN PRINT "UP"  
IF (JOYSTK(0) OR JOYSTK(1)) AND 16 THEN 150 ELSE 10
```

## **LINE INPUT "*prompt*"; *stringVar***

The LINE INPUT statement is similar to INPUT, but has the following differences:

- \* A question mark (?) is not displayed.
- \* The input data can only be assigned to one string variable.
- \* Commas, colons, quotation marks and leading spaces are all included in the string.

The *prompt* is optional.

```
LINE INPUT A$.  
LINE INPUT "READY> " ; C$
```

## **MAXVAL (*num1*, *num2*)**

The MAXVAL function compares the two numeric arguments and returns the larger value.

## **MID\$(*oldString*, *position*, *length*) = *newString***

The MID\$ assignment statement allows you to replace a portion of one string with the contents of another. *OldString* is the name of the string variable to be modified. *Position* specifies the position of the first character in *oldString* to be replaced.

The *length* argument is optional and specifies the number of characters to replace. If *length* is omitted then the computer uses either the length of *newString* or the number of positions remaining in *oldString*, whichever is smaller.

### **NOTES:**

If *length* is larger than the length of *newString* then all of *newString* is used.

The length of *oldString* never changes. The number of replacement characters is clipped to the number of positions remaining in *oldString*.

## **MINVAL (*num1*, *num2*)**

The MINVAL function compares the two numeric arguments and returns the smaller value.

## **ON BREAK...**

The ON BREAK statement permits trapping of the BREAK key so that your program can respond in some way other than stopping. There are three variations of ON BREAK:

ON BREAK CONT	Continue without stopping (disable BREAK).
ON BREAK GOTO <i>line</i>	Go to the specified line whenever BREAK is pressed.
ON BREAK STOP	Restore normal operation (stop when BREAK is pressed).

## ON ERROR...

The ON ERROR statement permits the trapping of errors so that your program can respond in some way other than stopping. There are three variations of ON ERROR:

ON ERROR CONT	Continue with the next statement instead of stopping.
ON ERROR GOTO <i>line</i>	Go to the specified line whenever an error occurs.
ON ERROR STOP	Restore normal operation (stop and report the error).

The ERROR statement can be used to simulate an error condition. The ERRN and ERRL functions can be used to determine the type of error and the line number in which it occurred.

## POS (*device*)

POS returns the current line position on the specified output device. The two output devices are the screen and a printer. Passing 0 as the argument to the POS function returns the current cursor position within a line on the screen. Passing -2 as the argument to the POS function returns the current column position of the printer's carriage.

## RENUM *new, start, interval*

Renumbers program lines and updates all line number references in statements such as GOTO, GOSUB, etc.

Renumbering starts with the first program line whose current line number is greater than or equal to *start*. If *start* is omitted then all of the program's lines will be renumbered.

The first new line number to be assigned is specified by *new*. If *new* is omitted then 10 will be used. Subsequent line numbers are assigned by adding the value of *interval*. If *interval* is omitted then a default interval of 10 is used.

RENUM	Renumber all lines. Use 10 for the first new number and interval.
RENUM 1, , 1	Renumber all lines. Use 1 for the first new number and interval.
RENUM 500, 100, 5	Renumbers lines 100 and beyond to 500, 505, 510...

## STRING\$ (*length, character*)

The STRING\$ function produces a string value from a repeated character. The *length* argument specifies how many times to repeat the character in order to produce the result. The *character* argument may be a number representing the ASCII code of the character or it can be a string from which the first character will be used.

A\$ = STRING\$( 32, "-" )	Assigns a string of 32 dashes to A\$.
PRINT STRING\$( 4, 13 )	Prints 4 carriage returns to the screen.

## **SWAP *var1, var2***

Exchanges the values of two variables. The variables must be of the same type (string or numeric). Individual elements within arrays may also be swapped.

```
SWAP J , K
SWAP A$ , B$
SWAP D$ ( N ) , D$ ( N+1 )
```

## **TIMER**

Returns the value of the built-in timer. The timer is set to zero when the computer is turned on and then begins incrementing once every sixtieth of a second (approximately). When the timer value reaches 65535, it starts over from 0.

**NOTE:** *The timer pauses during some operations including File I/O, Printing, RENUM and MERGE.*

## **TIMER = *number***

Sets the computer's TIMER value to *number*, which may be any integer from 0 to 65535.

## **TROFF**

Turns off program Tracing.

## **TRON**

Turns on program Tracing. Running a BASIC program while tracing is on will cause each line number to be printed to the screen as it is executed.

## **WAIT *milliseconds***

Causes the computer to pause for the specified number of milliseconds (0-65535).

## **WPEEK ( *address* )**

Returns the 16-bit word (0..65535) from memory where the MSB is read from *address* and the LSB from *address+1*.

## **WPOKE *address* , *word***

Stores the 16-bit word (0..65535) into memory with the MSB placed at *address* and the LSB at *address+1*.

# OTHER FEATURES AND CHANGES

## MACHINE RESTART

You can do a complete restart of the computer by holding down the SHIFT key while pressing the RESET button on the back of the computer.

## USING STOCK BASIC

You can startup the computer in stock Microcolor Basic by holding down the CONTROL key while switching on the power.

## KEYBOARD SHORTCUTS

The keyboard shortcuts for CSAVE (Ctrl-3) and CLOAD (Ctrl-4) have been changed to produce SAVE and LOAD (without the C).

Pressing SHIFT and BREAK together will clear the screen.

## HEX CONSTANTS

You can use hexadecimal constants prefixed with &H.

```
POKE &HBFFF, &H4C
```

## CLS 32 / CLS 40 / CLS 80 (Alice 32/90 only)

These CLS statements which change the screen mode may now be used in a program. They no longer reset the computer.

## RESTORE TO LINE

The RESTORE statement will accept an optional line number argument. This causes the system to start searching for DATA statements at the specified line number when the next READ statement is executed.

```
RESTORE 500
```

## VARPTR CORRECTION

The VARPTR function in MicroColor Basic was returning a signed integer result. This caused a negative value to be returned for variables located at addresses above \$7FFF. This problem has been fixed and VARPTR will now always return a positive number.

# ERROR CODES

Abbreviation	#	Description
NF	0	NEXT without FOR
SN	2	Syntax error
RG	4	RETURN without GOSUB
OD	6	Out of DATA
FC	8	Illegal Function Call
OV	10	Overflow
OM	12	Out of Memory
UL	14	Undefined Line number
BS	16	Bad Subscript
DD	18	Doubly Dimensioned array
D0	20	Division by Zero
ID	22	Illegal in Direct mode
TM	24	Type Mismatch
OS	26	Out of String space
LS	28	Length of String (too long)
ST	30	String formula is Too complex
CN	32	Can't continue
IO	34	Input/Output error
FM	36	Wrong File Mode
DN	38	Bad Device Number (or un-allocated file number)
NE	40	File does Not Exist
WP	42	Write Protected
FN	44	Badly formed File (or directory) Name
FS	46	File System error
IE	48	Input past End of file
FD	50	Unacceptable File Data
AO	52	File is Already Open
NO	54	File is Not Open
DS	56	Direct Statement in file
AE	58	File Already Exists
DF	60	Disk is Full