COMMODORE BASIC 3.5 MANUAL

version 2.1

by Janne Peräaho
(& Anders Persson)

COPYRIGHT

INTRODUCTION

Basic is a high level language which is based on the following six concepts: commands, statements, functions, variables, operators, and expressions.

Commands and statements are instructions to the computer to perform a certain task (for example an instruction to load a basic program into memory). The difference between them is that Basic commands are intented to be used in direct mode, while statements should be used in programs. However, in most cases commands can be used as statements in a program if you prefix them with a line number. You can also use several statements as commands by using them in direct mode (i.e. without line numbers).

A function performs a simple task, based on a given arguments, and it always replies with a value - a result.

Operators are used for calculations, for determining equalities/inequalities, and for logical operations. For example + is an operator used for addition.

Expressions are clauses composed of constants, variables, and/or operators. For example A+B*3 is a valid expression.

This manual's purpose is to provide detail information about presented Basic elements. I hope you find it useful.

MANUAL FORMAT

The Commodore BASIC 3.5 manual is divided into seven sections:

```
Commands    : the commands used for working with programs to edit,
              store, and erase them.
Functions   : the string, numeric, and print functions.
Operators   : the arithmetic and logical operators.
Statements  : the BASIC program statements used in numbered lines of
              programs.
Variables   : the different types of variables and legal variable
              names.
Basic errors: the error messages given by BASIC.
Disk errors : the error messages given by a disk drive.
              store, and erase them.
```

The items presented in sections follow consistent format convensions to make them as clear as possible. In most cases, there are several examples to illustrate what the actual command, function or statement looks like.

The syntax of items are described by using the following consepts:

KEYWORDS appear in uppercase letters. You must type keywords exactly as they appear!

ARGUMENTS appear within angle brackets. Arguments are parts that you select.

SQUARE BRACKETS ([]) show optional arguments. You select any or none of the arguments listed.

VERTICAL BAR (|) separates lists of options from which you can choose only one.

A SEQUENCE OF THREE DOTS (...) means that an option or argument can be repeated more than once.

QUOTATION MARKS ("") enclose character strings, file names, and other expressions. When arguments are enclosed in quotation marks in a format, you must include the quotation marks in your command, function, or statement. Quotation marks are required parts of a command, function or statement.

PARENTHESES (()). When arguments are enclosed in parentheses in a format, you must include the parentheses in your command, function, or statement.

COMMANDS

AUTO
BACKUP
COLLECT
CONT
COPY
DELETE
DIRECTORY
DLOAD
DSAVE
HEADER
HELP
KEY
LIST
LOAD
NEW
RENAME
RENUMBER
RUN
SAVE
SCRATCH
VERIFY

command/AUTO command/AUTO

NAME
AUTO -- Controls the automatic line numbering

ABBREVIATION
a <shift> U

SYNOPSIS
AUTO [<line>]

FUNCTION
Turns on the automatic line numbering feature which eases the job of entering programs by typing the line numbers for you. As you enter each program line and press <return> the next line number is printed on the screen, with the cursor in position to begin typing that line. AUTO with no argument turns off auto line numbering, as does RUN.

INPUTS
<line> - increment between line numbers

RESULT
With argument turns on automatic line numbering.
With no argument turns off auto line numering.

EXAMPLES
AUTO 10
Automatically numbers line in increments of ten.

AUTO 50
Automatically numbers line in increments of fifty.

AUTO
Turns off automatic line numbering.

NOTES
This statement is executable only in direct mode.

BUGS
None

command/BACKUP command/BACKUP

NAME
BACKUP -- Copies all the files on a disk to another disk

ABBREVIATION
    b <shift> A

SYNOPSIS
    BACKUP D<src_drive> TO D<trg_drive>[,ON U<unit>]

FUNCTION
    This command copies all the files on a disk to another disk on a dual drive system. You can copy onto a new disk without first using the HEADER command to format the new disk because the BACKUP command copies all the information on the disk, including the format. You should always BACKUP important disks in case the original is lost or damaged.
    Because the BACKUP command also HEADERS disks, it destroys any information on the disk onto which you're copying information. So if you're backing up onto a previously used disk, make sure it contains no programs you wish to keep.

INPUTS
    <src_drive> - source drive number
    <trg_drive> - target drive number
    <unit>   - target drive unit number

RESULT
    The contents of the source disk is copied to the target disk.

EXAMPLES
    BACKUP D0 TO D1
        Copies all files from the disk in drive 0 to the disk in drive 1.

    BACKUP DO TO D1, ON U9
        Copies all files from drive 0 to drive 1 in disk drive unit 9.

NOTES
    This command can only be used with dual disk drive.

BUGS
    -

command/COLLECT   command/COLLECT

NAME
    COLLECT -- Deletes references to improperly closed files

ABBREVIATION
    col <shift> L

SYNOPSIS
    COLLECT [D<drive>][,ON U<unit>]

FUNCTION
    Use this command to free up space allocated to improperly closed files and deletes references to these files from the directory.

INPUTS
    <drive> - target drive number
    <unit> - target drive unit number

RESULT
    Frees up disk space allocated to improperly closed files.

EXAMPLES
    COLLECT D0

NOTES
    None

BUGS
    None

    None

command/CONT   command/CONT

NAME

CONT -- Re-start the execution of a program that has been stopped

ABBREVIATION

c <shift> O

SYNOPSIS

CONT

FUNCTION

This command is used to re-start the execution of a program that has been stopped by either using the STOP statement, or an END statement within the program.

INPUTS

None

RESULT

The program will resume execution where it left off.

EXAMPLES

CONT

NOTES

CONT will not work if you have changed or added lines of the program (or even just moved the cursor to a program line and hit <return> without changing anything), if the program stopped due to an error, or if you caused an error before trying to re-start the program.

BUGS

None

command/COPY command/COPY

NAME

COPY -- Copies a file

ABBREVIATION

co <shift> P

SYNOPSIS

```
COPY [D<src_drive>,]"<src_file>" TO [D<trg_drive>,]"<trg_file>"
     [,ON U<unit>]
```

FUNCTION

Copies a file on the disk in one drive (the source file) to the disk in the other on dual disk drive only, or creates a copy of a file on the same drive (with a different file name).

INPUTS

```
<src_drive> - source drive number
<src_file>  - source file name
<trg_drive> - target drive number
<trg_file>  - target file name
<unit>      - target drive unit number
```

RESULT

A copy of a file is created.

EXAMPLES

```
COPY D0,"NOON" TO D1,"NIGHT"
    Copies NOON from drive 0 to drive 1, renaming it NIGHT.

COPY D0,"STUFF" TO D1,"STUFF"
    Copies STUFF from drive 0 to drive 1.

COPY D0 TO D1
    Copies all files from drive 0 to drive 1.

COPY "CATS" TO "DOGS"
    Copies CATS as a program called DOGS on the same drive.
```

NOTES

None

BUGS

None

NAME
    DELETE -- Deletes lines of BASIC text

ABBREVIATION
    de <shift> L

SYNOPSIS
    DELETE [<first_line>][-<last_line>]

FUNCTION
    Deletes lines of BASIC text.

INPUTS
    <first_line> - first line to be deleted
    <last_line> - last line to be deleted

RESULT
    Deletes lines of BASIC text.

EXAMPLES
    DELETE 75
        Deletes line 75.

    DELETE 10-50
        Deletes lines 10 through 50 inclusive.

    DELETE -50
        Deletes all lines from the beginning of the program up to and
        including line 50.

    DELETE 75-
        Deletes all lines from 75 on to the end of the program.

NOTES
    This command can be executed only in direct mode.

BUGS
    None

command/DIRECTORY command/DIRECTORY

NAME
    DIRECTORY -- Displays a disk directory

ABBREVIATION
    di <shift> R

SYNOPSIS
    DIRECTORY [D<drive>][,U<unit>][,"<file>"]

FUNCTION
    Displays a disk directory on the screen. Use <ctrl>-S to pause the
    display (any other key restarts the display after a pause). Use the C=
    key (the Commodore key) to slow it down.

INPUTS
    <drive> - drive number
    <unit> - drive unit number
    <file> - file name and/or pattern

RESULT
    Lists all files or files matching the given pattern.

EXAMPLES
    DIRECTORY
        List all files on the disk.

    DIRECTORY D1, U9, "WORK"
        Lists the file on disk drive unit 9 (8 is default), drive 1, named
        WORK.

    DIRECTORY "AB"
        Lists all files starting with the letters "AB", like ABOVE, ABOARD,
        etc.

DIRECTORY D0, "FILE ?.BAK"
The ? is a wild-card that matches any single character in that position: FILE 1.BAK, FILE 2.BAK, FILE 3.BAK all match the string.

NOTES

The DIRECTORY command cannot be used to print a hard copy. You must load the disk directory (destroying the program currently in memory) to do that.

To print out the DIRECTORY of drive 0, unit 8, use the following:

```
LOAD"$0",8
OPEN4,4:CMD4:LIST
PRINT#4:CLOSE4
```

BUGS

None

None

command/DLOAD command/DLOAD

NAME

DLOAD -- Loads a program from disk into a memory

ABBREVIATION

d <shift> L

SYNOPSIS

DLOAD "<file>"[,D<drive>][,U<unit>]

FUNCTION

This command loads a program from disk into a memory. (Use LOAD to load programs on tape.) You must supply a file name.

INPUTS

<file> - file name and/or pattern
<drive> - drive number
<unit> - drive unit number

RESULT

A program is loaded from disk into a memory.

EXAMPLES

DLOAD "DTRUCK"
Searches the disk for the program "DTRUCK" and LOADs it.

DLOAD (A$)
LOADs a program from disk whose name is in the variable A$. You will get an error if A$ is empty.

NOTES

The DLOAD command can be used within a BASIC program to find and RUN another program on disk. This is called chaining.

BUGS

None

command/DSAVE command/DSAVE

NAME

DSAVE -- Stores a program on disk

ABBREVIATION

d <shift> S

SYNOPSIS

DSAVE "<file>"[,D<drive>][,U<unit>]

FUNCTION

This command stores a program on disk. (Use SAVE to store programs on tape.) You must supply a file name.

INPUTS

<file> - file name and/or pattern
<drive> - drive number

<unit> - drive unit number

RESULT

A program is stored on a disk.

EXAMPLES

DSAVE "DDAY"

SAVEs the program "DDAY" to disk.

DSAVE (A$)

SAVEs to disk program whose name is in the variable A$.

DSAVE "PROG 3",D0,U9

SAVEs the program "PROG 3" to the disk drive with a unit number of 9

NOTES

None

BUGS

None

command/HEADER command/HEADER

NAME

HEADER -- Formats a disk

ABBREVIATION

he <shift> A

SYNOPSIS

HEADER "<diskname>",D<drive>[,I<id>][,ON U<unit>]

FUNCTION

Before you can use a new disk for the first time you must format it with the HEADER command. If you want to erase an entire disk for re-use you can use the HEADER command. This command divides the disk into sections called blocks, and it creates a table of contents, called a directory or catalog, on the disk. The diskname can be any name up to 16 characters long. The id number is any 2 characters. Give each disk a unique id number. Be careful when you HEADER a disk because the HEADER command erases all stored data. Giving no id number allows you to perform a quick header. The old id number is used. You can only use the quick header method if the disk was previously formatted, since the quick header only cleans out the directory rather than formatting the disk.

INPUTS

<diskname> - name for the disk (max length 16 characters)
<drive> - drive number
<id> - disk identification number (max length 2 characters)
<unit> - drive unit number

RESULT

A ready to use empty disk.

EXAMPLES

HEADER "MYDISK",I23,D0

HEADER "THEBALL",I45,D1,U8

NOTES

None

BUGS

None

command/HELP command/HELP

NAME

HELP -- Displays the erroneous program line

ABBREVIATION

None

SYNOPSIS

HELP

```
FUNCTION
    The HELP command is used after you get an error in your program. When
    you type HELP, the line where the error occured is listed, with the
    portion containing the error displayed in flashing characters.

INPUTS
    None

RESULT
    Displays the line which has caused the last error. The portion
    containing the error is displayed in flashing characters.

EXAMPLES
    HELP

NOTES
    None

BUGS
    None


    None

command/KEY                                                     command/KEY

NAME
    KEY -- Assigns a string into a function key

ABBREVIATION
    k <shift> E

SYNOPSIS
    KEY [<key>,<string>]

FUNCTION
    There are eight (8) function keys available to the user on your
    Commodore 16 computer: four unshifted and four shifted. Your Commodore
    16 allows you to define what each key does when pressed. KEY without any
    parameter specified gives a listing displaying all the current KEY
    assignments. The data you assign to a key is typed out when that
    function key is pressed. The maximum length for all the definitions
    together is 128 characters. Entire commands (or a series of commands)
    can be assigned to a key.

INPUTS
    <key>    - function key number (1-8)
    <string> - string to be assigned into a key

RESULT
    Shows current function key bindings or assigns a string into a function
    key.

EXAMPLES
    KEY 7,"GRAPHICS0"+CHR$(13)+"LIST"+CHR$(13)
        Causes the computer to select text mode and list your program
        whenever the "F7" key is pressed (in direct mode). The CHR$(13) is
        the ASCII character for <return>.

NOTES
    Use CHR$(34) to incorporate a double quote into a KEY string.
    The keys may be redefined in a program. For Example:

        10 KEY2,"TESTING"+CHR$(34):KEY3,"NO"

    To define function keys as they are on the Commodore 64 and VIC 20:

        10 FOR I=1 TO 8:KEY I,CHR$(I+132):NEXT

    To restore all function keys to their default values, reset your
    Commodore 16 by turning it off and on, or press the RESET button.

BUGS
    None


    None

command/LIST                                                   command/LIST
```

NAME
LIST -- Lets you look at lines of a BASIC program

ABBREVIATION
l <shift> I

SYNOPSIS
LIST [<first_line>][-[<last_line>]]

FUNCTION
The LIST command lets you look at lines of a BASIC program that have been typed or LOADed into the computer's memory. When LIST is used alone (without any numbers following it), you get a complete LISTing of the program on your screen, which may be slowed down by holding the C= key (Commodore key), paused by <ctrl>-S (unpaused by pressing any other key), or STOPed by pressing the <run/stop> key. If you follow the word LIST with a line number, your computer only shows that line number. If you type LIST with two numbers separated by a dash, the computer shows all lines from the first to the second line number. If you type LIST followed by a number and just a dash, it shows all the lines from that number to the end of the program. And if you type LIST, a dash, and then a number, you get all the lines from the beginning of the program to that line number. Using these variations, you can examine any portion of a program, or easily bring lines to the screen for modification.

INPUTS
<first_line> - first BASIC line to be shown
<last_line> - last BASIC line to be shown

RESULT
Brings BASIC program lines to the screen.

EXAMPLES
LIST
Shows entire program.

LIST 100-
Shows from line 100 until the end of the program.

LIST 10
Shows only line 10.

LIST -100
Shows lines from the beginning until line 100.

LIST 10-200
Shows lines from 10 to 200, inclusive.

NOTES
None

BUGS
None

None

command/LOAD command/LOAD

NAME
LOAD -- Loads a program from storage device into a memory

ABBREVIATION
l <shift> O

SYNOPSIS
LOAD ["<file>"[,<device>][,<rel_flag>]]

FUNCTION
This is the command to use when you want to use a program stored on tape or on disk. If you type just LOAD and hit the <return> key the computer screen goes blank. Press play, and the computer starts looking for a program on the tape. When it finds one, the computer prints "FOUND <filename>". You can hit the C= key (Commodore key) to LOAD; if you don't press the key, the computer resumes searching on the tape after a brief interval. Once the program is LOADed, you can RUN, LIST, or change it.
You can also type the word LOAD followed by a program name, which is most often a name in quotes ("<program_name>"). The name may be followed by a comma (outside of the quotes) and a number (or numeric variable),

    which acts as a device number to determine where the program is stored
    (disk or tape). If there is no number given, your computer assumes
    device number 1.
    The LOAD command can be used within a BASIC program to find and RUN the
    next program on tape. This is called chaining.
    The relocate flag (<rel_flag>) determines where in memory a program is
    loaded. A relocate flag of 0 tells the computer to load the program at
    the start of the BASIC program area, and a flag of 1 tells it to LOAD
    from the point where it was SAVEd. The default value of the relocate
    flag is 0.

INPUTS
    <file>   - file name and/or pattern to be loaded
    <device>   - storage device number
    <rel_flag> - relocate flag (0 or 1)

RESULT
    A program is loaded from storage device into a memory.

EXAMPLES
    LOAD
        Reads in the next program on tape.

    LOAD "BASES"
        Searches tape for a program called BASES, and LOADS it if it is
        found.

    LOAD A$
        Looks for a program whose name is in the variable called A$.

    LOAD "BRIDGES",8
        Looks for the program called BRIDGES on the disk drive, and LOADs it
        if found.

NOTES
    Device 1: Tape.
    Device 8: Disk.

    Relocate flag of 1 is generally used only when loading machine language
    programs.

BUGS
    None

command/NEW   command/NEW

NAME
    NEW -- Erases BASIC program in memory

ABBREVIATION
    None

SYNOPSIS
    NEW

FUNCTION
    This command erases the entire program in memory and clears out any
    variables that may have been used. Unless the program was stored
    somewhere, it is lost until you type it in again. Be careful when you
    use this command.
    The NEW command can also be used as a statement in a BASIC program. When
    your computer gets to this line, the program is erased and everything
    stops. This is not especially useful under normal circumstances.

INPUTS
    None

RESULT
    BASIC program is erased from memory and all variables are cleared out.

EXAMPLES
    NEW

NOTES
    None

BUGS
    None

```
command/RENAME                                                 command/RENAME

NAME
    RENAME -- Renames a file

ABBREVIATION
    re <shift> N

SYNOPSIS
    RENAME [D<drive>,]"<old_filename>" TO "<new_filename>"[,U<unit>]

FUNCTION
    Used to rename a file on a disk.

INPUTS
    <drive>        - drive number
    <old_filename> - original file name
    <new_filename> - new file name
    <unit>         - drive unit number

RESULT
    Renamed file.

EXAMPLES
    RENAME D0,"ASSET" TO "LIABILITY"
        Changes the name of the file from ASSET to LIABILITY.

NOTES
    None

BUGS
    None


    None

command/RENUMBER                                             command/RENUMBER

NAME
    RENUMBER -- Renumbers program lines

ABBREVIATION
    ren <shift> U

SYNOPSIS
    RENUMBER [<new_line>[,<increment>[,<start_line>]]]

FUNCTION
    This command renumbers BASIC program lines beginning from the first line
    (set as 10) renumbering in increments of 10 at the end of the program.
    You can supply starting line (<start_line>), spacing between line
    numbers (<increment>), and/or first line number (<new_line>).
    The first line number is the number of the first line in the program
    after renumbering (default is 10). The increment is the spacing between
    line numbers, i.e. 10, 20, 30 etc. (It also defaults to 10.). The first
    line number is the line number in the program where renumbering is to
    begin. This allows you to renumber a portion of your program. It
    defaults to the first line of your program.

INPUTS
    <new_line>   - line number which replaces the start line number
                   (<start_line>). Default line number is 10.
    <increment>  - spacing between line numbers (default is 10)
    <start_line> - line number where renumbering starts (default is the
                   first line)

RESULT
    Renumbered program line(s).

EXAMPLES
    RENUMBER 20,20,1
        Starting at line 1, renumbers the program. Line 1 becomes line 20,
        and other lines are numbered in increments of 20.

    RENUMBER ,,65
        Starting at line 65, renumbers in increments of 10. Line 65 becomes
        line 10 (unless there are already lines numbered 10-64, in which
```

case the command is not carried out).

NOTES
    This command can only be executed from direct mode.

BUGS
    None

    None

command/RUN command/RUN

NAME
    RUN -- Executes a program

ABBREVIATION
    r <shift> U

SYNOPSIS
    RUN [<line>]

FUNCTION
    Once program has been typed into memory or LOADed, the RUN command makes it start working. RUN clears all variables in the program before starting program execution. If there is no number following the command RUN, the computer starts with the lowest numbered program line. If there is a number following the RUN command execution starts at that line.

INPUTS
    <line> - line number where program execution should start

RESULT
    BASIC program is executed.

EXAMPLES
    RUN
        Starts program working from lowest line number.

    RUN 100
        Starts program at line 100.

NOTES
    RUN may be used within a program.

BUGS
    None

command/SAVE command/SAVE

NAME
    SAVE -- Stores program in a storage device

ABBREVIATION
    s <shift> A

SYNOPSIS
    SAVE [<file>[,<device>[,<eot_flag>]]]

FUNCTION
    This command stores a program currently in memory onto a tape or disk. If you just type the word SAVE and press <return>, your computer attempts to store the program on the tape. It has no way of checking if there is already a program on the tape in that location, so be careful with your tapes. If you type SAVE command followed by a name in quotes or a string variable name, the computer gives the program that name, so it may be more easily located and retrieved in the future. If you want to specify a device number for the SAVE, follow the name by a comma (after the quotes) and a number or numeric variable. After the number on a tape command, there can be a comma and a second number (0 or 1). If the second number is 1, the computer puts an END-OF-TAPE marker (<eot_flag>) after your program. If you are trying to LOAD a program and the computer finds one of these markers rather than the program you are trying to LOAD, you get a FILE NOT FOUND ERROR.

INPUTS
    <file>   - file name
    <device>   - storage device number

```
    <eot_flag> - end-of-tape flag (0 or 1)

RESULT
    The program currently in memory is stored in a storage device.

EXAMPLES
    SAVE
        Stores program to tape without a name.

    SAVE "MONEY"
        Stores on tape with name MONEY.

    SAVE A$
        Stores on tape with name in variable A$.

    SAVE "YOURSELF",8
        Stores on disk with name YOURSELF.

    SAVE "GAME",1,1
        Stores on tape with name GAME and places an END-OF-TAPE marker after
        the program.

NOTES
    Device 1: tape drive.
    Device 8: disk drive.

BUGS
    None


command/SCRATCH                                               command/SCRATCH

NAME
    SCRATCH -- Deletes a file from disk

ABBREVIATION
    sc <shift> R

SYNOPSIS
    SCRATCH "<file>"[,D<drive>][,U<unit>]

FUNCTION
    Deletes a file from the disk directory. As a precaution, you are asked
    "Are you sure?" before your computer completes the operation. Type a Y
    to perform the SCRATCH or type N to cancel the operation. Use this
    command to erase unwanted files, to create more space on the disk.

INPUTS
    <file>  - file name and/or pattern to be deleted
    <drive> - drive number
    <unit>  - drive unit number

RESULT
    File is erased from the disk directory.

EXAMPLES
    SCRATCH "MY BACK",D1
        Erases the file MY BACK from the disk in drive 1.

NOTES
    None

BUGS
    None


command/VERIFY                                                 command/VERIFY

NAME
    VERIFY -- Checks stored program against the one in memory

ABBREVIATION
    v <shift> E

SYNOPSIS
    VERIFY "<file>"[,<device>[,<rel_flag>]]

FUNCTION
```

This command causes your computer to check the program on tape or disk against the one in memory. This is proof that the program you just SAVEd is really saved, to make sure that nothing went wrong. This command is also very useful to position a tape so that your computer resumes writing following the end of the last program on the tape. All you do is tell the computer to VERIFY the name of the last program on the tape. It will do so, and tell you that the programs don't match (which you already knew). Now the tape is where you want it, and you can store the next program without fear of erasing an old one.
VERIFY without anything after the command causes the computer to check the next program on tape, regardless of its name, against the program now in memory. VERIFY followed by a program name (in quotes) or a string variable searches the tape for that program and then checks its. VERIFY followed by a name and a comma and a number checks the program on the device with that number. The relocate flag (<rel_flag>) is the same as in the LOAD command.

INPUTS

<file>   - file name and/or pattern to be checked
<device>   - storage device number
<rel_flag> - relocate flag (0 or 1)

RESULT

Verification.

EXAMPLES

VERIFY
Checks the next program on the tape.

VERIFY "REALITY"
Searches for REALITY on tape, checks against memory.

VERIFY "ME",8,1
Searches for ME on disk, then checks.

NOTES

Device 1: tape.
Device 8: disk.

BUGS

None

FUNCTIONS

¶
ABS
ASC
ATN
CHR$
COS
DEC
ERR$
EXP
FN
FRE
HEX$
INSTR
INT
JOY
LEFT$
LEN
LOG
MID$
PEEK
POS
RCLR
RDOT
RGR
RIGHT$
RLUM
RND

SGN
SIN
SPC
SQR
STR$
TAB
TAN
USR
VAL

function/¶ function/¶

NAME
¶ -- Returns the value of pi

ABBREVIATION
None

SYNOPSIS
¶(<dummy>)

FUNCTION
The pi symbol, when used in an equation, has the value 3.14159265.

INPUTS
<dummy> - dummy argument and can be any value

RESULT
3.14159265 (numeric).

EXAMPLES
None

NOTES
None

BUGS
None

None

function/ABS function/ABS

NAME
ABS -- Returns the magnitude of the numeric value

ABBREVIATION
a <shift> B

SYNOPSIS
ABS(<number>)

FUNCTION
The absolute value function returns the magnitude of the argument <number>.

INPUTS
<number> - numeric value

RESULT
Magnitude of the given number (numeric).

EXAMPLES
None

NOTES
None

BUGS
None

None

function/ASC function/ASC

NAME

ASC -- Returns character's ASCII code

ABBREVIATION
a <shift> S

SYNOPSIS
ASC(<string>)

FUNCTION
This function returns the ASCII code (number) of the first character of <string>.

INPUTS
<string> - string

RESULT
ASCII code number of the first character of the given string (numeric).

EXAMPLES
None

NOTES
None

BUGS
None

function/ATN function/ATN

NAME
ATN -- Returns arctangent

ABBREVIATION
a <shift> T

SYNOPSIS
ATN(<number>)

FUNCTION
Returns the angle whose tangent is <number>, measured in radians.

INPUTS
<number> - tangent (number)

RESULT
Angle measured in radians (numeric).

EXAMPLES
None

NOTES
None

BUGS
None

None

function/CHR$ function/CHR$

NAME
CHR$ -- Returns a character in the base of ASCII code

ABBREVIATION
c <shift> H

SYNOPSIS
CHR$(<ascii_code>)

FUNCTION
This function returns a string character whose ASCII code is <ascii_code>.

INPUTS
<ascii_code> - character's ASCII code (0-255)

RESULT

Character corresponding the given ASCII code (string).

EXAMPLES

```
PRINT CHR$(65);CHR$(66);CHR$(67)
    ABC
```

NOTES
None

BUGS
None

function/COS function/COS

NAME
COS -- Returns cosine value

ABBREVIATION
None

SYNOPSIS
COS(<angle>)

FUNCTION
Returns the value of the cosine of <angle>, where <angle> is an angle measured in radians.

INPUTS
<angle> - angle in radians

RESULT
Cosine value of an angle (numeric).

EXAMPLES
None

NOTES
None

BUGS
None

function/DEC function/DEC

NAME
DEC -- Converts hexadecimal number to decimal

ABBREVIATION
None

SYNOPSIS
DEC(<string>)

FUNCTION
Returns decimal value of hexadecimal-string.

INPUTS
<string> - hexadecimal string (0000-FFFF)

RESULT
Decimal value of the given hexadecimal number (numeric).

EXAMPLES

```
N=DEC("F4")
```

NOTES
None

BUGS
None

None

function/ERR$ function/ERR$

NAME
    ERR$ -- Returns string describing error condition

ABBREVIATION
    e <shift> R

SYNOPSIS
    ERR$(<err_condition>)

FUNCTION
    This function returns string describing given error condition (<err_condition>).

INPUTS
    <err_condition> - error condition number

RESULT
    Error message (string).

EXAMPLES
    None

NOTES
    None

BUGS
    None

function/EXP   function/EXP

NAME
    EXP -- Raises constant e to the given power

ABBREVIATION
    e <shift> X

SYNOPSIS
    EXP(<power>)

FUNCTION
    Returns the value of the mathematical constant e (2.71828183) raised to the power of <power>.

INPUTS
    <power> - power (number)

RESULT
    Raises constant e to the given power.

EXAMPLES
    None

NOTES
    None

BUGS
    None

function/FN   function/FN

NAME
    FN -- Calls user-defined function

ABBREVIATION
    None

SYNOPSIS
    FN<fnc_name>(<number>)

FUNCTION
    Returns the value of the user-defined function <fnc_name> created in a DEF FN statement.

INPUTS
    <fnc_name> - name of the user-defined function
    <number>   - value to be passed to the function

RESULT
Returns the result of the called function (numeric).

EXAMPLES
None

NOTES
None

BUGS
None

function/FRE function/FRE

NAME
FRE -- Returns the amount of available memory

ABBREVIATION
f <shift> R

SYNOPSIS
FRE(<dummy>)

FUNCTION
This function returns the number of unused bytes available in memory.

INPUTS
<dummy> - dummy argument and can be any value

RESULT
Amount of free memory in bytes.

EXAMPLES
None

NOTES
None

BUGS
None

None

function/HEX$ function/HEX$

NAME
HEX$ -- Converts a decimal number into a hexadecimal one

ABBREVIATION
h <shift> E

SYNOPSIS
HEX$(<number>)

FUNCTION
This function returns a 4 character string containing the hexadecimal representation of value <number>.

INPUTS
<number> - value to be evaluated (0-65535)

RESULT
Hexadecimal representation of the given decimal value (string).

EXAMPLES
None

NOTES
None

BUGS
None

function/INSTR function/INSTR

NAME
    INSTR -- Searches for a substring

ABBREVIATION
    in <shift> S

SYNOPSIS
    INSTR(<string_1>,<string_2>[,<start_pos>])

FUNCTION
    Returns position of string <string_2> in string <string_1> at or after the starting-position (<start_pos>). The starting-position defaults to the beginning of string <string_2>. If no match is found, a value of 0 is returned.

INPUTS
    <string_1> - string to be searched
    <string_2> - string to search
    <start_pos> - position where searching should start

RESULT
    Returns position of the second string in the first string (numeric). If the string was not found, returns 0.

EXAMPLES
    PRINT INSTR("THE CAT IN THE HAT","CAT")
        The result is 5, because CAT starts at the fifth character in the first string.

NOTES
    None

BUGS
    None

    None

function/INT   function/INT

NAME
    INT -- Extracts the integer portion of a decimal number

ABBREVIATION
    None

SYNOPSIS
    INT(<number>)

FUNCTION
    Returns the integer portion of <number>, with all decimal places to the right of the decimal point removed. The result is always less-than or equal to <number>. Thus, any negative numbers with decimal places become the integer less-than their current value (e.g. INT(-4.5)=-5).

INPUTS
    <number> - number to be evaluated

RESULT
    Integer part of a given number (numeric).

EXAMPLES
    X=INT(X*100+.5)/100
        Rounds to the next highest penny.

NOTES
    If the INT function is to be used for rounding off, the form is INT(<number>+.5) or INT(<number>-.5).

BUGS
    None

    None

function/JOY   function/JOY

NAME
    JOY -- Polls joystick port

ABBREVIATION
    j <shift> O

SYNOPSIS
    JOY(<port>)

FUNCTION
    This function returns the state of joystick connected to port <port>. Any value returned of 128 or more means the fire button is also depressed. The direction is indicated as follows:

```
                UP       FIRE

                1        128

             8     2

        LEFT   7    0    3   RIGHT

             6     4

                5

               DOWN
```

INPUTS
    <port> - joystick port number (1-2)

RESULT
    State of joystick (numeric).

EXAMPLES
    100 J=JOY(2)
        If value of 135 returned, joystick in port 2 has turned to left with fire button.

NOTES
    None

BUGS
    None

    None

function/LEFT$ function/LEFT$

NAME
    LEFT$ -- Strips string from the right

ABBREVIATION
    le <shift> F

SYNOPSIS
    LEFT$(<string>,<length>)

FUNCTION
    This function returns a string containing the leftmost <length> characters of string <string>.

INPUTS
    <string> - source string
    <length> - number of characters to be included in result string

RESULT
    String containing leftmost <length> characters of the string <string>.

EXAMPLES
    None

NOTES
    None

BUGS
    None

function/LEN function/LEN

NAME
    LEN -- Returns the number of characters in the string

ABBREVIATION
    None

SYNOPSIS
    LEN(<string>)

FUNCTION
    This function returns the number of characters (including spaces and
    other symbols) in the string <string>.

INPUTS
    <string> - string to be evaluated

RESULT
    Number of characters (numeric).

EXAMPLES
    None

NOTES
    None

BUGS
    None

    None

function/LOG   function/LOG

NAME
    LOG -- Returns the natural log of the given number

ABBREVIATION
    None

SYNOPSIS
    LOG(<number>)

FUNCTION
    This function returns the natural log of <number>. The natural log is
    log to the base e.

INPUTS
    <number> - number to be evaluated

RESULT
    Natural log of the given number.

EXAMPLES
    None

NOTES
    To convert to log base 10, divide by LOG(10).

BUGS
    None

function/MID$ function/MID$

NAME
    MID$ -- Returns a substring

ABBREVIATION
    m <shift> I

SYNOPSIS
    MID$(<string>,<start_pos>,<length>)

FUNCTION
    This function returns a string containing <length> characters, starting
    from the <start_pos> character in string <string>. MID$ can also be used
    on the left side of assignment statement as a pseudo-variable as well as
    a function.

An error results if <start_pos>+<length> is greater than the length of the source string (<string>).

INPUTS
```
<string>    - source string
<start_pos> - starting position of the substring
<length>    - length of the substring to be extracted or length of the
              target area
```

RESULT

A string, which length is <length>, extracted from the source string (<string>) at the given position (<start_pos>).

EXAMPLES

Using MID$ as a pseudo-variable:

```
10 A$="THE LAST GOODBYE"
20 PRINT A$
30 MID$(A$,6,3)="ONG"
40 PRINT A$
    THE LAST GOODBYE
    THE LONG GOODBYE
```

Using MID$ for extracting substring:

```
10 PRINT MID$("THE LAST GOODBYE",10,4)
    GOOD
```

NOTES

None

BUGS

None

function/PEEK function/PEEK

NAME

PEEK -- Gives contents of memory location

ABBREVIATION

p <shift> E

SYNOPSIS

PEEK(<address>)

FUNCTION

This function gives the contents of memory location <address>, where <address> is located in the range of 0 to 65535, returning a result from 0 to 255. This is often used in conjunction with the POKE statement.

INPUTS

<address> - memory location (0-65535)

RESULT

Contents of the memory location (numeric).

EXAMPLES

```
PEEK(1024)
```

NOTES

None

BUGS

None

function/POS function/POS

NAME

POS -- Current cursor x position

ABBREVIATION

None

SYNOPSIS

POS(<dummy>)

FUNCTION
This function returns the number of the column (0-39) where the next PRINT statement begins on the screen.

INPUTS
<dummy> - dummy argument and can be any value

RESULT
Cursor x position (numeric).

EXAMPLES
None

NOTES
None

BUGS
None

None

function/RCLR function/RCLR

NAME
RCLR -- Returns color source's current color

ABBREVIATION
r <shift> C

SYNOPSIS
RCLR(<color_src>)

FUNCTION
This function returns current color assigned to source <color_src>.

INPUTS
<color_src> - color source (0-4):
0 - background
1 - foreground
2 - multicolor 1
3 - multicolor 2
4 - border

RESULT
Returns current color: 1-16 (numeric).

EXAMPLES
None

NOTES
None

BUGS
None

function/RDOT function/RDOT

NAME
RDOT -- Returns information about the current PC location

ABBREVIATION
r <shift> D

SYNOPSIS
RDOT(<info_flag>)

FUNCTION
This function returns information about the current position of the pixel cursor (PC) at XPOS/YPOS.

INPUTS
<info_flag> - required information:
0 - current pixel cursor x position
1 - current pixel cursor y position
2 - color source used at current PC position

RESULT

Returns PC's current x position, y position, or color source used at current PC position (numeric).

EXAMPLES
    None

NOTES
    None

BUGS
    None

    None

function/RGR   function/RGR

NAME
    RGR -- Returns current graphic mode

ABBREVIATION
    r <shift> G

SYNOPSIS
    RGR(<dummy>)

FUNCTION
    This function returns current graphic mode.

Mode	Description
0	normal text
1	high-resolution graphics
2	high-resolution graphics, split screen
3	multicolor graphics
4	multicolor graphics, split screen

INPUTS
    <dummy> - dummy argument and can be any value

RESULT
    Current graphic mode (numeric).

EXAMPLES
    None

NOTES
    None

BUGS
    None

function/RIGHT$ function/RIGHT$

NAME
    RIGHT$ -- Strips string from the left

ABBREVIATION
    r <shift> I

SYNOPSIS
    RIGHT$(<string>,<length>)

FUNCTION
    This function returns a string containing the right-most <length> characters of string <string>.

INPUTS
    <string> - source string
    <length> - number of characters to be included in result string

RESULT
    String containing right-most <length> characters of the string <string>.

EXAMPLES
    None

NOTES

```
    None

BUGS
    None



function/RLUM                                           function/RLUM

NAME
    RLUM -- Returns color source's current luminance

ABBREVIATION
    r <shift> L

SYNOPSIS
    RLUM(<color_src>)

FUNCTION
    This function returns current luminance level assigned to color source
    <color_src>.

INPUTS
    <color_src> - color source (0-4):
                  0 - background
                  1 - foreground
                  2 - multicolor 1
                  3 - multicolor 2
                  4 - border

RESULT
    Returns current luminance: 0-7 (numeric).

EXAMPLES
    None

NOTES
    None

BUGS
    None



function/RND                                             function/RND

NAME
    RND -- Generates a random number

ABBREVIATION
    r <shift> N

SYNOPSIS
    RND(<seed>)

FUNCTION
    This function returns a random number between 0 and 1. This is useful in
    games, to simulate dice rolls and other elements of change, and is also
    used in some statistical applications. The first random number should be
    generated by the formula RND(-TI), to start things off differently every
    time. After this, the number in <seed> should be a 1, or any positive
    number. If <seed> is zero, RND is re-seeded from the hardware clock ever
    time RND is used. A negative value for <seed> seeds the random number
    generator using <seed> and gives a random number sequence. The use of
    the same negative number for <seed> as a seed results in the same
    sequence of random numbers. A positive value gives random numbers based
    on the previous seed.

INPUTS
    <seed> - a seed, or what the random number is based on

RESULT
    A random number between 0 and 1 (numeric).

EXAMPLES
    100 X=INT(RND(1)*6)+INT(RND(1)*6)+2
        Simulates two dice.

    100 X=INT(RND(1)*1000)+1
        Number from 1-1000.
```

```
100 X=INT(RND(1)*150)+100
    Number from 100 to 249.
```

NOTES

To simulate the rolling of a die, use the formula INT(RND(1)*6+1). First the random number from 0-1 is multiplied by 6, which expands the range to 0-6 (actually, greater than zero and less than six). Then 1 is added, making the range 1 to under 7. The INT function chops off all the decimal places, leaving the result as a digit from 1 to 6.
To simulate 2 dice, add two of the numbers obtained by the above formula together.

BUGS

None

None

function/SGN function/SGN

NAME

SGN -- Returns number's sign

ABBREVIATION

s <shift> G

SYNOPSIS

SGN(<number>)

FUNCTION

This function returns the sign, as in positive, negative, or zero, of <number>. The result is:

```
+1 if <number> is positive
 0 if <number> is zero
-1 if <number> is negative
```

INPUTS

<number> - number to be evaluated

RESULT

Number's sign. -1 is returned if number was negative, 0 if number was zero, or 1 if number was positive (numeric).

EXAMPLES

None

NOTES

None

BUGS

None

function/SIN function/SIN

NAME

SIN -- Returns sine value

ABBREVIATION

s <shift> I

SYNOPSIS

SIN(<angle>)

FUNCTION

This is the trigonometric sine function. The result is the sine of <angle>, where <angle> is an angle in radians.

INPUTS

<angle> - angle in radians

RESULT

Sine value of an angle (numeric).

EXAMPLES

<example_function_call>

NOTES
None

BUGS
None

function/SPC function/SPC

NAME
SPC -- Skips over spaces

ABBREVIATION
s <shift> P

SYNOPSIS
SPC(<skip>)

FUNCTION
This function is used in the PRINT statement to skip over <skip> spaces.

INPUTS
<skip> - number of spaces to be skipped (0-255)

RESULT
Skips over <skip> spaces in the PRINT statement.

EXAMPLES
None

NOTES
None

BUGS
None

function/SQR function/SQR

NAME
SQR -- Returns the square root

ABBREVIATION
s <shift> Q

SYNOPSIS
SQR(<number>)

FUNCTION
This function returns the square root of <number>, where <number> is a positive number or 0. If <number> is negative, an ILLEGAL QUANTITY ERROR results.

INPUTS
<number> - number to be evaluated

RESULT
Square root of the given number (numeric).

EXAMPLES
None

NOTES
None

BUGS
None

None

function/STR$ function/STR$

NAME
STR$ -- Converts number into a string

ABBREVIATION
st <shift> R

SYNOPSIS
    STR$(<number>)

FUNCTION
    This function converts a decimal number into a string.

INPUTS
    <number> - number to be converted

RESULT
    A string corresponding a given numeric value (string).

EXAMPLES

```
10 A=10.5
20 PRINT A
30 A$=STR$(A)
40 PRINT A$
    10.5
    10.5
```

NOTES
    None

BUGS
    None

function/TAB   function/TAB

NAME
    TAB -- Sets cursor's x position

ABBREVIATION
    t <shift> A

SYNOPSIS
    TAB(<column>)

FUNCTION
    This function is used in the PRINT statement. The next item to be printed is in column number <column>.

INPUTS
    <column> - cursor's x position (0-39)

RESULT
    Sets cursor to the given column within the PRINT statement.

EXAMPLES
    None

NOTES
    None

BUGS
    None

function/TAN   function/TAN

NAME
    TAN -- Returns tangent value

ABBREVIATION
    None

SYNOPSIS
    TAN(<angle>)

FUNCTION
    This function gives the tangent of <angle>, where <angle> is an angle in radians.

INPUTS
    <angle> - angle in radians

RESULT

Tangent value of an angle (numeric).

EXAMPLES
None

NOTES
None

BUGS
None

function/USR function/USR

NAME
USR -- Executes a machine language program with a parameter

ABBREVIATION
u <shift> S

SYNOPSIS
USR(<parameter>)

FUNCTION
When this function is used, the program jumps to a machine language program whose starting point is contained in memory locations 1281 (lower byte of the 16 bit memory address) and 1282 (higher byte of the 16 bit memory address). The parameter <parameter> is passed to the machine language program in the floating point accumulator. Another number is passed back to the BASIC program (by the machine language program) through the calling variable. In other words, this allows you to exchange a variable between machine code and BASIC.

INPUTS
<parameter> - numeric value to be passed to the machine language program

RESULT
USR calls a machine language program with a given numeric parameter. While exiting machine language program passes a another number back to the BASIC.

EXAMPLES
None

NOTES
I don't exactly know how the machine language program passes the value back to the BASIC. Maybe there is a special variable for this? I don't know.

BUGS
None

function/VAL function/VAL

NAME
VAL -- Converts string into a number

ABBREVIATION
None

SYNOPSIS
VAL(<string>)

FUNCTION
This function converts the string (<string>) into a number, and is essentially the inverse operation from STR$. The string is examined from the left-most character to the right, for as many characters as are in recognizable number format. If the computer finds illegal characters, only the portion of the string up to that point is converted.

INPUTS
<string> - string containing a number

RESULT
Number corresponding the number given in string (numeric).

EXAMPLES

```
10 X=VAL("123.456")
    X=123.456

10 X=VAL("3E03")
    X=3000

10 X=VAL("12A13B")
    X=12

10 X=VAL("RIUO17*")

    X=0

10 X=VAL("-1.23.23.23")
    X=-1.23
```

NOTES
    None

BUGS
    None

OPERATORS

The arithmetic operators include the following signs:

```
+ addition
- subtraction
* multiplication
/ division
^ raising to a power (exponentation); ^ = up arrow
```

On a line containing more tha one operator, there is a set order in which operations always occur. If several operators are used together, the computer assigns priorities as follows: First, exponentiation, then multiplication and division, and last, addition and subtraction. If two operations have the same priority, then calculations are performed in order from left to right. If you want these operations to occur in a different order, BASIC allows you to give a calculation a higher priority by placing parentheses around it. Operations enclosed in parentheses will be calculated before any other operation. You have to make sure that your equations have the same number of left parentheses as right parentheses, or you will get a SYNTAX ERROR message when your program is run.
There are also operators for equalities and inequalities, called relational operators. Arithmetic operators always take priority over relational operators.

```
=  equal to
<  less than
>  greater than
<= less than or equal to
=< less than or equal to
>= greater than or equal to
=> greater than or equal to
<> not equal to
>< not equal to
```

Finally there are thee logical operators, with lower priority than both arithmetic and relational operators:

```
AND
OR
NOT
```

These are used most often to join multiple formulas in IF...THEN statements. When they are used with arithmetic operators, they are evaluated last (i.e., after + and -).

Examples:

```
IF A=B AND C=D THEN 100
    Requires both A=B & C=D to be true.
```

```
IF A=B OR C=D THEN 100
    Allows either A=B or C=D to be true.

A=5:B=4:PRINT A=B
    Displays a value of 0.

A=5:B=4:PRINT A>B
    Displays a value of -1.

PRINT 123 AND 15:PRINT 5 OR 7
    Displays 11 and 7.
```

STATEMENTS

BOX
CHAR
CIRCLE
CLOSE
CLR
CMD
COLOR
DATA
DEF
DIM
DO
DRAW
END
FOR
GET
GET#
GETKEY
GOSUB
GOTO
GRAPHIC
GSHAPE
IF
INPUT
INPUT#
LET
LOCATE
MONITOR
NEXT
ON
OPEN
PAINT
POKE
PRINT
PRINT USING
PRINT#
PUDEF
READ
REM
RESTORE
RESUME
RETURN
SCALE
SCNCLR
SOUND
SSHAPE
STOP
SYS
TRAP
TROFF
TRON
VOL
WAIT

statement/BOX statement/BOX

NAME
    BOX -- Draws a rectangle

ABBREVIATION
    b <shift> O

SYNOPSIS
```
    BOX [<color_src>],<left>,<top>[,<right>,<bottom>][,<angle>[,<fill_flag>]
```

FUNCTION
    This command allows you to draw a rectangle of any size anywhere on the screen. To get the default value, include a comma without entering a value. Rotation is based on the centre of the rectangle. The Pixel Cursor (PC) is left at <right>,<bottom> after the BOX statement is executed.

INPUTS
```
    <color_src> - draw color source (0-3); default is 1 (foreground color)
    <left>      - scaled corner coordinate
    <top>       - scaled corner coordinate
    <right>     - scaled corner coordinate
    <bottom>    - scaled corner coordinate
    <angle>     - box rotation in clockwise degrees; default is 0 degrees
    <fill_flag> - fill flag (0 or 1); default is 0 (no filling)
```

RESULT
    Draws a rectangle.

EXAMPLES
```
    BOX 1,10,10,60,60
        Draws the outline of a rectangle.

    BOX ,10,10,60,60,45,1
        Draws a filled, rotated box (a diamond).

    BOX ,30,90,,45,1
        Draws a filled, rotated polygon.
```

NOTES
    None

BUGS
    None

statement/CHAR statement/CHAR

NAME
    CHAR -- Prints string on a screen

ABBREVIATION
    ch <shift> A

SYNOPSIS
```
    CHAR [<color_src>],<left>,<top>,"<string>"[,<reverse_flag>]
```

FUNCTION
    Text (alphanumeric strings) can be displayed on any screen at a given location by the CHAR command. Character data is read from the computer character ROM area. You supply the left (<left>) and top (<top>) coordinates of the starting position and the text string (<string>) you want to display, color (<color>) and reverse imaging (<reverse_flag>) are optional.
    The string is continued on the next line if it attempts to print past the right edge of the screen. When Used in TEXT mode, the string printed by the CHAR command works just like a PRINT string, including reverse field, cursors, flash on/off, etc. These control functions inside the string do not work when the CHAR command is used to display text in GRAPHIC mode.

INPUTS
```
    <color_src>    - printing color source (0-3)
    <left>         - character column (0-39)
    <top>          - character row (0-24)
    <string>       - text to be printed
    <reverse_flag> - reverse field flag (0=off, 1=on)
```

RESULT
    Prints given string on a screen at a given position.

EXAMPLES

```
CHAR 1,10,10,"HELLO!"
```

NOTES
    None

BUGS
    None

statement/CIRCLE   statement/CIRCLE

NAME
    CIRCLE -- Draws a circle, ellipse, arc, triangle or an octagon

ABBREVIATION
    c <shift> I

SYNOPSIS

```
CIRCLE [<color_src>][,<x>,<y>],<x_radius>[,[<y_radius>][,[<s_angle>]
       [,[<e_angle>][,[<rotation>][,<degrees>]]]]]
```

FUNCTION
    With the CIRCLE command you can draw a circle, ellipse, arc, triangle or an octagon. The final coordinate (Pixel Cursor location) is on the circumference of the circle at the ending arc angle. Any rotation (<rotation>) is about the centre. Arcs are drawn from the starting angle (<s_angle>) clockwise to the ending angle (<e_angle>). The segment increment (<degrees>) controls the coarseness of the shape, with lower values for inc creating rounder shapes.

INPUTS

```
<color_src> - draw color source (0-3)
<x>         - scaled centre x-coordinate (defaults to Pixel Cursor, PC)
<y>         - scaled centre y-coordinate (defaults to Pixel Cursor, PC)
<x_radius>  - scaled x radius
<y_radius>  - scaled y radius (defaults  to <x_radius>)
<s_angle>   - starting arc angle (default 0)
<e_angle>   - ending arc angle (default 360)
<rotation>  - rotation in clockwise degrees (default is 0 degrees)
<degrees>   - degrees between segments (default is 2 degrees)
```

RESULT
    Draws a circle, ellipse, arc, triangle or an octagon.

EXAMPLES

```
CIRCLE,160,100,65,10
    Draws an ellipse.

CIRCLE,160,100,65,50
    Draws an oval.

CIRCLE,60,40,20,18,,,,45
    Draws an octagon.

CIRCLE,260,40,20,,,,,90
    Draws a diamond.

CIRCLE,60,140,20,18,,,,120
    Draws a triangle.
```

NOTES
    None

BUGS
    None

statement/CLOSE   statement/CLOSE

NAME
    CLOSE -- Closes an open logical file

ABBREVIATION

cl <shift> O

SYNOPSIS
    CLOSE <file>

FUNCTION
    This command completes and closes any files used by OPEN statements.

INPUTS
    <file> - file number to be closed

RESULT
    Closes an open logical file.

EXAMPLES
    CLOSE 2
        Logical file 2 is closed.

NOTES
    None

BUGS
    None

statement/CLR   statement/CLR

NAME
    CLR -- Erases any variables in memory

ABBREVIATION
    c <shift> L

SYNOPSIS
    CLR

FUNCTION
    This command erases any variables in memory, but leaves the program itself intact. This command is automatically executed when a RUN or NEW command is given, or when any editing is performed.

INPUTS
    None

RESULT
    Erases any variables in memory.

EXAMPLES
    CLR

NOTES
    None

BUGS
    None

statement/CMD   statement/CMD

NAME
    CMD -- Redirects output

ABBREVIATION
    c <shift> M

SYNOPSIS
    CMD <l_file>[,<w_list>]

FUNCTION
    CMD sends the output which normally would go to the screen (i.e. PRINT statement, LISTs, but not POKEs into the screen) to another device instead. This could be a printer, or a data file on tape or disk. This device or file must be OPENed first. The CMD command must be followed by a number or numeric variable referring to the file (<l_file>).

INPUTS
    <l_file> - logical file number
    <w_list> - UNKNOWN ARGUMENT!

RESULT

Redirects output.

EXAMPLES

```
10 OPEN 1,4
20 CMD 1
30 LIST
40 PRINT#1
50 CLOSE 1
```

Line 10: OPENs device number 4, which is the printer.
Line 20: All normal output now goes to the printer.
Line 30: The LISTing goes to the printer, not the screen - even the word READY.
Line 40: Set output back to the screen.
Line 50: Close the file.

NOTES

None

BUGS

None

None

statement/COLOR statement/COLOR

NAME

COLOR -- Assigns a color to the color source

ABBREVIATION

co <shift> L

SYNOPSIS

COLOR <color_src>,<color>[,<luminance>]

FUNCTION

Assigns a color to one of the 5 color sources:

Number	Source
0	background
1	foreground
2	multicolor 1
3	multicolor 2
4	border

Colors you can use are in the range 1-16 (1 is black, 2 is white, 9 is orange, etc. from your keyboard color keys). As an option, you can include the luminance level 0-7, with 0 being lowest and 7 being highest. Luminance defaults to 7. Luminance lets you select from eight levels of brightness for any color exept black.

INPUTS

<color_src> - color source (0-4)
<color> - color (1-16)
<luminance> - luminance (0-7)

RESULT

Assigns a color to the color source.

EXAMPLES

```
COLOR 1,1
```

NOTES

None

BUGS

None

None

statement/DATA statement/DATA

NAME

DATA -- Declares data items

ABBREVIATION

d <shift> A

SYNOPSIS

DATA <item>[[,<item>][,<...>[,<item>]]]

FUNCTION

This statement is followed by a list of items to be used by READ statements. The items may be numbers or words, and are separated by commas. Words need not be inside of quote marks, unless they contain any of the following characters: space, colon, or comma. If two commas have nothing between them, the value will be READ as a zero for a number, or an empty string. The DATA statement must be part of a program, otherwise it will not be recognized. Also see the RESTORE statement, which allows your computer to reread data.

INPUTS

<item> - constant which will be declared as a data item

RESULT

Declares data items to be read by READ command.

EXAMPLES

```
DATA 100,200,FRED,"WILMA",,3,14,ABC123
```

NOTES

None

BUGS

None

statement/DEF   statement/DEF

NAME

DEF FN -- Defines a function

ABBREVIATION

d <shift> E

SYNOPSIS

DEF FN <fnc_name>(<variable>)=<expression>

FUNCTION

This command allows you to define a complex calculation as a function. In the case of a long formula that is used several times within a program, this can save a lot of space.
The name you give the numeric function begins with the letters FN, followed by any legal numeric variable name (<fnc_name>). First you must define the function by using the statement DEF followed by the name (<fnc_name>) you've given the function. Following the name is a set of parentheses () with a numeric variable (<variable>) enclosed. Then you have an equal sign, followed by the formula (<expression>) you want to define. You can call the formula, substituting any number for a variable (<variable>).

INPUTS

<fnc_name>   - name of the function
<variable>   - variable name used in the formula
<expression> - formula

RESULT

Defines a function to be used within a program.

EXAMPLES

```
10 DEF FNA(X)=12*(34.75-X/.3)+X
20 PRINT FNA(7)
```

The number 7 is inserted each place X is located in the formula given in the DEF statement.

NOTES

DEF FN can only be used with standard numeric functions, not integer or string functions.

BUGS

None

None

NAME
    DIM -- Presents and reserves memory for an array

ABBREVIATION
    d <shift> I

SYNOPSIS

```
DIM <variable>(<subscripts>)[,<variable>(<subscripts>)][,<...>
    [,<variable>(<subscripts>)]]
```

FUNCTION
    Before you can use an array of variables, the program must first execute a DIM statement to establish the DIMensions of that array (unless there are 11 or fewer elements in the array). The statement DIM is followed by the name of the array (<variable>), which may be any legal variable name. Then, enclosed in parentheses, you put the number (or numeric variable) of elements (<subscripts>) in each dimension. An array with more than one dimension is called a matrix. You may use any number of dimensions, but keep in mind that the whole list of variables you are creating takes up space in memory, and it is easy to run out of memory if you get carried away. To figure the number of variables created with each DIM, multiply the total number of elements in each dimension of the array.
    You can dimension more than one array in a DIM statement by separating the arrays by commas. If the program executes a DIM statement for any array more than once, you'll get re'DIMed array error message. It is good programming practice to place DIM statements near the beginning of the program.

INPUTS
    <variable>   - array name (legal variable name)
    <subscripts> - number of elements in an array

RESULT
    Presents and reserves memory for an array or arrays.

EXAMPLES

```
DIM A$(40),B7(15),CC%(4,4,4)
        !      !       !
        !      !       +- 125 Elements
        !      +---------  16 Elements
        +----------------  41 Elements
```

NOTES
    Each array starts with element 0.
    Integer (single-digit) arrays take up 2/5ths of the space of floating point arrays.

BUGS
    None

    None

statement/DO statement/DO

NAME
    DO -- Defines a program loop

ABBREVIATION

```
DO    None
EXIT  None
LOOP  lo <shift> O
UNTIL  u <shift> N
WHILE  w <shift> H
```

SYNOPSIS

```
DO [UNTIL <bool_arg>|WHILE <bool_arg>] <statements> [EXIT]
LOOP [UNTIL <bool_arg>|WHILE <bool_arg>]
```

FUNCTION
    Performs the statements between the DO statement and the LOOP statement. If no UNTIL or WHILE modifies either the DO or the LOOP statement, execution of the intervening statements continues indefinitely. If an EXIT statement is encountered in the body of a DO loop, execution is transferred to the first statement following the LOOP statement. DO loops may be nested, following the rules defined for FOR-NEXT loops.
    If the UNTIL parameter is used, the program continues looping until the

```
    boolean argument is satisfied (becomes TRUE). The WHILE parameter is
    basically the opposite of the UNTIL parameter: the program continues
    looping as long as the boolean argument is TRUE.

INPUTS
    <bool_arg>   - boolean argument. For example A=1 or H>=57
    <statements> - statements to be executed

RESULT
    Performs the statements between the DO statement and the LOOP statement
    forever or until WHILE or UNTIL condition is satisfied.

EXAMPLES
    DO WHILE A$="":GETA$:LOOP

NOTES
    None

BUGS
    None



statement/DRAW                                                    statement/DRAW

NAME
    DRAW -- Draws dots, lines, and shapes

ABBREVIATION
    d <shift> R

SYNOPSIS
    DRAW [<color_src>][<x>,<y>,][[,]TO <x>,<y>][,<...>[,<x>,<y>]]

FUNCTION
    With this command you can draw individual dots, lines, and shapes. You
    supply color source (<color_src>), starting and ending points (<x>,<y>).

INPUTS
    <color_src> - draw color source (0-3); default is 1 (foreground color)
    <x>         - scaled x coordinate
    <y>         - scaled y coordinate

RESULT
    Draws dots, lines, or shapes.

EXAMPLES
    DRAW 1,100,50
        Draws a dot.

    DRAW ,10,10, TO 100,60
        Draws a line.

    DRAW TO 25,30
        Draws a line.

    DRAW ,10,10 TO 100,60 TO 10,10
        Draws a shape.

NOTES
    None

BUGS
    None



statement/END                                                      statement/END

NAME
    END -- Stops program execution

ABBREVIATION
    e <shift> N

SYNOPSIS
    END

FUNCTION
    When the program executes an END statement, the program stops RUNing
```

immediately. You may use the CONT command to restart the program at the statement following the END statement.

INPUTS
None

RESULT
Program stops running.

EXAMPLES
```
END
```

NOTES
None

BUGS
None

statement/FOR statement/FOR

NAME
FOR -- Defines a program loop

ABBREVIATION
f <shift> O

SYNOPSIS
```
FOR <loop_var>=<start_val> TO <end_val> [STEP <increment>]
```

FUNCTION
This statement works with the NEXT statement to set up a section of the program that repeats for a set number of times. You may just want your computer to count up to a large number so the program pauses for a few seconds, in case you need something counted, or something must be done a certain number of times (such as printing).
The loop variable (<loop_var>) is the variable that is added to or subtracted from during the FOR-NEXT loop. The start value (<start_val>) and the end value (<end_val>) are the beginning and ending counts for the loop variable.
The logic of the FOR statement is as follows. First, the loop variable (<loop_var>) is set to the start value (<start_val>). When the program reaches a line with the command NEXT, it adds the STEP increment (<increment>) to the value of the loop variable and checks to see if it is higher than the end of loop value. If it is not higher, the next line executed is the statement immediately following the FOR statement. If the loop variable is larger than the end of loop number, then the next statement executed is the one following the NEXT statement.
The end loop value may be followed by the word STEP and another number or variable. This allows you to count backwards, by fractions, or any way necessary.

INPUTS
```
<loop_var>  - variable which holds the loop counter value
<start_val> - start value for loop variable (<loop_var>)
<end_val>   - end value for loop variable (<loop_var>)
<increment> - value to be added to or subtracted from loop variable
```

RESULT
Performs the statements between the FOR statement and the NEXT statement until the loop variable reaches the end value.

EXAMPLES
```
10 FOR L=1 TO 20
20 PRINT L
30 NEXT L
40 PRINT "BLACKJACK! L="L
```
Prints the numbers from one to twenty ob the screen, followed by the message BLACKJACK! L=21.

```
10 FOR L=1 TO 100
20 FOR A=5 TO 11 STEP 2
30 NEXT A
40 NEXT L
```
FOR-NEXT loop with loop variable A is nested inside the larger one.

NOTES
STEP increment default value is 1.
A STEP value can be positive or negative.

You can set up loops inside one another. This is known as nesting loops. You must be careful to nest loops so that the last loop to start is the first one to end.

BUGS
None

statement/GET statement/GET

NAME
GET -- Gets data from the keyboard

ABBREVIATION
g <shift> E

SYNOPSIS
GET <variable>

FUNCTION
The GET statement is a way to get data from the keyboard one character at a time. When the GET is executed, the character that was typed is received. If no character was typed, then a null (empty) character is returned, and the program continues without waiting for a key. There is no need to press the <return> key, and in fact the <return> key can be received with a GET.
The word GET is followed by a variable name, usually a string variable. If a numeric ware used and any key other than a number was hit, the program would stop with an error message. The GET statement may also be put into a loop, checking for an empty result, which waits for a key to be struck to continue. The GETKEY statement could also be used in this case.

INPUTS
<variable> - acquired data will be stored in this variable

RESULT
Data acquired from the keyboard is stored in the target variable (<variable>).

EXAMPLES
```
10 GET A$:IF A$ <> "A" THEN 10
    This line waits for the "A" key to be pressed to continue.
```

NOTES
This command can only be executed within a program.

BUGS
None

statement/GET# statement/GET#

NAME
GET# -- Gets data from a file or a device

ABBREVIATION
None

SYNOPSIS
GET# <file>,<variable>

FUNCTION
Used with a previously OPENed device or file to input one character at a time. Otherwise, it works like the GET statement.

INPUTS
<file> - file/device number to be read
<variable> - acquired data will be stored in this variable

RESULT
Data acquired from the file/device is stored in the target variable (<variable>).

EXAMPLES
```
10 GET#1,A$
```

NOTES

This command can only be executed within a program.

BUGS

None

statement/GETKEY   statement/GETKEY

NAME

GETKEY -- Gets data from the keyboard

ABBREVIATION

getk <shift> E

SYNOPSIS

GETKEY <variable>

FUNCTION

The GETKEY statement is vary similar to the GET statement. Unlike the GET statement, GETKEY waits for the user to type a character on the keyboard. This lets it to be used easily to wait for a single character to be typed.

INPUTS

<variable> - acquired data will be stored in this variable

RESULT

Data acquired from the keyboard is stored in the target variable (<variable>).

EXAMPLES

```
10 GETKEY A$
```

This line waits for a key to be struck. Typing any key will continue the program.

NOTES

This command can only be executed within a program.

BUGS

None

statement/GOSUB   statement/GOSUB

NAME

GOSUB -- Calls a subroutine

ABBREVIATION

go <shift> S

SYNOPSIS

GOSUB <line>

FUNCTION

This statement is like the GOTO statement, exept that your computer remembers where it came from. When a line with a RETURN statement is encountered, the program jumps back to the statement immediately following the GOSUB. The target of a GOSUB statement is called a subroutine. A subroutine is useful if there is a routine in your program that can be used by several different portions of the program. Instead of duplicating the section of program over and over, you can set it up as a subroutine, and GOSUB to it from the different parts of the program

INPUTS

<line> - line number where subroutine begins

RESULT

Program execution continues in a given subroutine (<line>) until RETURN statement is encountered.

EXAMPLES

```
20 GOSUB 800
...
800 PRINT "HI THERE":RETURN
```

Line 20 means: go to the subroutine beginning at line 800 and execute it.

NOTES

None

BUGS

None

statement/GOTO statement/GOTO

NAME

GOTO -- Redirects program execution

ABBREVIATION

g <shift> O

SYNOPSIS

GOTO <line>
GO TO <line>

FUNCTION

After a GOTO or GO TO statement is executed, the next line to be executed will be the one with the line number following the word GOTO. When used in direct mode, GOTO <line> allows you to start execution of the program at the given line number without clearing the variables.

INPUTS

<line> - line number where program execution should continue

RESULT

Program execution continues at the given line.

EXAMPLES

```
10 PRINT"REPETITION IS THE MOTHER OF LEARNING"
20 GOTO 10
```

The GOTO in line 20 causes line 10 to be run continuously, until the <run/stop> key is pressed.

NOTES

None

BUGS

None

statement/GRAPHIC statement/GRAPHIC

NAME

GRAPHIC -- Changes graphic mode

ABBREVIATION

g <shift> R

SYNOPSIS

GRAPHIC <mode>[,<clr_flag>]

FUNCTION

This statement puts your computer in one of its 5 graphic modes:

Mode	Description
0	normal text
1	high-resolution graphics
2	high-resolution graphics, split screen
3	multicolor graphics
4	multicolor graphics, split screen

When executed, GRAPHIC (mode 1,2,3 or 4) allocates a 10KB bit-mapped area, and the BASIC text area is moved down below the hi-res area. This area remain allocated even if the user returns to TEXT mode (GRAPHIC 0). If 1 is given in the GRAPHIC statement as the second argument, the screen is also cleared.

INPUTS

<mode> - graphic mode (0-4)
<clr_flag> - screen clear flag (0=off, 1=on)

RESULT

Changes graphic mode and clears screen if clear flag is on.

EXAMPLES
    GRAPHIC 1,1
        Selects hi-res graphic mode and clears the screen.

    GRAPHIC 4,0
        Selects multicolor graphics with an area for text, without clearing
        the screen.

NOTES
    None

BUGS
    None

statement/GSHAPE   statement/GSHAPE

NAME
    GSHAPE -- Displays a shape on a graphic screen

ABBREVIATION
    g <shift> S

SYNOPSIS
    GSHAPE <shape>[,[<x>,<y>][,<mode>]]

FUNCTION
    A rectangular graphic clips can be displayed on a multicolor or high resolution graphics screen by the GSHAPE statement. If you type GSHAPE with the shape variable (<shape>) the shape will be drawn with the top left of the shape positioned at the pixel cursor. The shape variable may be followed by a graphic coordinates (<x> and <y>) and a replacement mode value (<mode>). The coordinates tell where the shape should be drawn on the screen and the mode value how it should be drawn.
    There are five possible replacement mode values:

Mode	Description
0	place shape as is (default)
1	place field inverted shape
2	OR shape with area
3	AND shape with area
4	XOR shape with area

    In mode 0 the shape is drawn to the graphic screen as it is. In this mode shape overwrites completely the graphic area where it is drawn.
    In mode 1 the shape overwrites the graphic area just like in mode 0 but this time the overwriting shape is inverted.
    In mode 2 logical operation OR is executed with the shape data and the bit map to be replaced (the graphic area). Result is a transparent shape on top of the bit map.
    In mode 3 logical operation AND is executed with the shape data and the bit map to be replaced. Result is a shape filtered bit map.
    In mode 4 logical operation XOR is executed with the shape data and the bit map to be replaced. Result is a shape filtered bit map.

INPUTS
    <shape> - string variable containing a shape to be drawn
    <x>   - scaled x coordinate. The default display position is the PC (pixel cursor)
    <y>   - scaled y coordinate. The default display position is the PC (pixel cursor)
    <mode> - replacement mode (0-4)

RESULT
    Displays a shape on a graphic screen.

EXAMPLES
    GSHAPE V$,,,1
        Displays V$ shape with background and foreground colors reversed, with the top left of the shape positioned at the pixel cursor (PC).

NOTES
    None

BUGS
    None

statement/IF statement/IF

NAME

IF -- Conditional execution

ABBREVIATION

None

SYNOPSIS

IF <expression> THEN <clause> [:ELSE <clause>]

FUNCTION

IF-THEN lets the computer analyze a BASIC expression preceded by IF and take one of two possible courses of action. If the expression is true, the statement following THEN is executed. This expression may be any BASIC statement. If the expression is false, the program goes directly to the next line, unless an ELSE clause is present. The expression being evaluated may be a variable or formula, in which case it is considered true if nonzero, and false if zero. In most cases, there is an expression involving relational operators (=, <, >, <=, >=, <>, AND, OR, NOT).

The ELSE clause, if present, must be in the same line as the IF-THEN part. When an ELSE clause is present, it is executed when the THEN clause isn't executed. In other words, the ELSE clause executes when the IF expression is FALSE.

INPUTS

<expression> - condition (BASIC expression resulting true or false value

<clause> - statements to be executed

RESULT

If expression (<expression>) is true, statements following the word THEN will be executed and if expression is false, statements following the wo ELSE will be executed. If ELSE is not present, program goes directly to the next line.

EXAMPLES

```
                           THEN
                        +-------+
                        ! Print !
                        !  "OK" !
                       /+-------+
                  Yes /
                     /
            IF      /
         +--------+ /
X --------! Check: !/
         !  X>0?  !\
         +--------+ \
                     \
                      \     ELSE
                   No \ +-------+
                       \!       !
                        !  End  !
                        +-------+
```

50 IF X>0 THEN PRINT"OK":ELSE END

Checks the value of X. If X is greater than 0, the THEN clause is executed, and the ELSE clause isn't. If X is not greater than 0, the ELSE clause is executed and the THEN clause isn't.

NOTES

None

BUGS

None

None

statement/INPUT statement/INPUT

NAME

INPUT -- Asks input from the user and stores acquired data

ABBREVIATION

None

SYNOPSIS

INPUT["<prompt>";]<variable>[,<...>,<variable>]

FUNCTION

The INPUT statement allows the computer to ask for data from the person running the program and place it into a variable or variables. The program stops, prints a question mark (?) on the screen, and waits for the person to type the answer and press the <return> key.
The word INPUT is followed by a variable name (<variable>) or list of variable names separated by commas. There may be a message inside quotes before the list of variables to be input (<prompt>). If this message (called a prompt) is present, there must be a semicolon (;) after the closing quote of the prompt. When more than one variable is to be INPUT, they should be separated by commas when typed in. If not, the computer asks for the remaining values by printing two question marks (??). If you press <return> key without INPUTting values, the INPUT variables retain the values previously held for those variables.

INPUTS

<prompt> - prompt string
<variable> - acquired data will be stored in this variable

RESULT

Asks input from the user and stores acquired data in the target variable(s).

EXAMPLES

```
10 INPUT "WHAT'S YOUR NAME";A$
20 INPUT "AND YOUR FAVOURITE COLOR";B$
30 INPUT "WHAT'S THE AIR SPEED OF A SWALLOW";A
```

NOTES

This statement can only be executed within a program.

BUGS

None

statement/INPUT# statement/INPUT#

NAME

INPUT# -- Reads data from a file or a device

ABBREVIATION

i <shift> N

SYNOPSIS

INPUT#<file>,<variable>[,<...>,<variable>]

FUNCTION

This works like INPUT, but takes the data from a previously OPENed file or device. No prompt string is allowed.

INPUTS

<file> - file/device number to be read
<variable> - acquired data will be stored in this variable

RESULT

Reads data from the file/device and stores acquired data in the target variable(s).

EXAMPLES

```
10 INPUT#2,A$,C,D$
```

NOTES

This statement can only be executed within a program.

BUGS

None

statement/LET statement/LET

NAME

LET -- Sets a value to a variable

ABBREVIATION

l <shift> E

SYNOPSIS
    LET <variable>=<expression>

FUNCTION
    The word LET is hardly ever used in programs, since it is not necessary, but the statement itself is the heart of all BASIC programs. Whenever a variable is defined or given a value, LET is always implied. The variable name which is to get the result of a calculation is on the left side of the equal sign, and the number or formula is on the right side.

INPUTS
    <variable>   - name of the target variable
    <expression> - number or formula to be stored in variable (<variable>)

RESULT
    Given value is stored to the given variable.

EXAMPLES
```
10 LET A=5
20 B=6
30 C=A*B+3
40 D$="HELLO"
```
        LET is implied (but not necessary) in lines 20, 30, and 40.

NOTES
    None

BUGS
    None

    None

statement/LOCATE   statement/LOCATE

NAME
    LOCATE -- Changes pixel cursor position

ABBREVIATION
    lo <shift> C

SYNOPSIS
    LOCATE <x>,<y>

FUNCTION
    The LOCATE command lets you put the pixel cursor (PC) anywhere on the screen. The PC is the current location of the starting point of the next drawing. Unlike the regular cursor, you can't see the PC, but you can move it with the LOCATE command.
    You can find out where the PC is at any time by using the RDOT(0) function to get the x-coordinate and RDOT(1) to get the y-coordinate. The color source of the dot at the PC can be found by printing RDOT(2). (In all drawing commands where a color option is available, you may select a value from 0 to 3, corresponding to the background, foreground, multicolor 1, or multicolor 2 as the color source.)

INPUTS
    <x> - scaled x coordinate
    <y> - scaled y coordinate

RESULT
    Puts the pixel cursor at the given position.

EXAMPLES
```
LOCATE 160,100
```
        Positions the PC in the centre of the high resolution screen.

NOTES
    None

BUGS
    None

statement/MONITOR   statement/MONITOR

NAME
    MONITOR -- Starts machine language monitor

ABBREVIATION
    m <shift> O

SYNOPSIS
    MONITOR

FUNCTION
    This command takes you out of BASIC into the build-in machine language monitor program. The monitor is used to develop, debug, and execute machine language programs more easily than from BASIC.

INPUTS
    None

RESULT
    Starts machine language monitor.

EXAMPLES
    MONITOR

NOTES
    When in the monitor, typing an "X" and pressing <return> gets you back to BASIC. Also read the "tedmon.pdf" document for a description of all the monitor commands.

BUGS
    None

statement/NEXT   statement/NEXT

NAME
    NEXT -- Completes a FOR loop

ABBREVIATION
    n <shift> E

SYNOPSIS
    NEXT [<variable>[,<...>,<variable>]]

FUNCTION
    The NEXT statement is used with the FOR statement. When the computer encounters a NEXT statement, it goes back to the corresponding FOR statement and check the loop variable. If the loop is finished, execution proceeds with the statement after the NEXT statement. The word NEXT may be followed by a variable name, a list of variable names separated by commas, or no variable names. If there are no names listed, the last loop started is the one being completed. If the variables are given, they are completed in order from left to right.

INPUTS
    <variable> - name of the FOR loop variable

RESULT
    Causes computer to go back to the corresponding FOR statement and check the FOR loop variable. Depending on loop variable value NEXT either exits the loop or repeats it once more.

EXAMPLES
```
10 FOR L=1 TO 10:NEXT
20 FOR L=1 TO 10:NEXT L
30 FOR L=1 TO 10:FOR M=1 TO 10:NEXT M,L
```

NOTES
    None

BUGS
    None

statement/ON   statement/ON

NAME
    ON -- Redirects program execution conditionally

ABBREVIATION
    None

SYNOPSIS

ON <expression> GOSUB <line>[,<...>,<line>]
ON <expression> GOTO <line>[,<...>,<line>]

FUNCTION

This command can make the GOTO and GOSUB statements into special versions of the IF statement. The word ON is followed by a formula, then either GOTO or GOSUB, and a list of line numbers separated by commas. If the result of the calculation of the formula (<expression>) is 1, the first line (<line>) in the list is executed. If the result is 2, the second line number is executed, and so on. If the result is 0, or larger than the number of line numbers in the list, the next line executed is the statement following the ON statements. If the number is negative, an ILLEGAL QUANTITY ERROR results.

INPUTS

<expression> - BASIC expression resulting a numeric value
<line> - line number where program execution should continue

RESULT

Program execution continues at the line chosen from the line number list according to a value determined by a BASIC expression (<expression>).

EXAMPLES

```
10 INPUT X:IF X<0 THEN 10
20 ON X GOTO 50, 30, 30, 70
25 PRINT "FELL THROUGH":GOTO 10
30 PRINT "TOO HIGH":GOTO 10
50 PRINT "TOO LOW":GOTO 10
70 END
```

When X = 1, ON sends control to the first line number in the list (50). When X = 2, ON sends control to the second line (30), etc.

NOTES

None

BUGS

None

statement/OPEN statement/OPEN

NAME

OPEN -- Opens a logical file for I/O operations

ABBREVIATION

o <shift> P

SYNOPSIS

OPEN <file>[,<device>[,<address>[,"<command>,<type>,<mode>"]]]

FUNCTION

The OPEN statement allows your computer to access devices such as the tape and disk for data, a printer, or even the screen. The word OPEN is followed by a logical file number (<file>), which is the number to which all other BASIC statements will refer. This number is from 1 to 255. There is normally a second number after the first called the device number (<device>). Device number 0 is the keyboard, 1 is the tape (default), 3 is the screen, 4 is the printer, 8 is usually the disk. A zero (0) may be included in front of the device number digit (e.g. 08 for 8). Following the second number may be a third number called the secondary address (<address>). In the case of the tape, this can be 0 for read, 1 for write, and 2 for write with end-of-tape marker at the end. In the case of the disk, the number refers to the channel number. In the printer, the secondary addresses are used to set the mode of the printer. There may also be a string following the third number, which could be a command to the disk drive or name of the file on tape or disk (<command>). The type (<type>) and mode (<mode>) refer to disk files only. (File types are prg, seq, rel, and usr; modes are read and write.)

INPUTS

<file> - logical file number for the file to be opened (1-255)
<device> - input/output device number
<address> - secondary address for device
<command> - command for device
<type> - file type (prg/seq/rel/usr)
<mode> - I/O mode (read/write)

RESULT

Opens a logical file for I/O operations.

EXAMPLES

```
10 OPEN 3,3
    OPENs the screen as a device.

10 OPEN 1,0
    OPENs the keyboard as a device.

10 OPEN 1,1,0,"UP"
    OPENs the tape for reading, file to be searched for is named UP.

OPEN 4,4
    OPENs a channel to use the printer.

OPEN 15,8,15
    OPENs the command channel on the disk.

5 OPEN 8,8,12,"TESTFILE,SEQ,WRITE"
    Creates a sequential disk file for writing.
```

NOTES

None

BUGS

None

statement/PAINT statement/PAINT

NAME

PAINT -- Fills an area with color

ABBREVIATION

```
p <shift> A
```

SYNOPSIS

```
PAINT [<color_src>][,[<x>,<y>][,<mode>]]
```

FUNCTION

The PAINT command lets you fill an area with color. It fills in the area around the specified point until a boundary of the same color (or any non-background color, depending on which mode you have chosen) is encountered. The final position of the Pixel Cursor (PC) will be at the starting point (<x>,<y>).

INPUTS

```
<color_src> - fill color source (0-3); default is 1 (foreground color)
<x>         - scaled x coordinate (starting point)
<y>         - scaled y coordinate (starting point)
<mode>      - fill mode (0 = paint an area defined by the color source
              selected; 1 = paint an area defined by any non-background
              color source)
```

RESULT

Fills in the area around the specified point until a boundary of the same color (or any non-background color, depending on which mode you have chosen) is encountered.

EXAMPLES

```
10 CIRCLE,160,100,65,50
20 PAINT,160,100
    Draws outline of circle and fills in the circle with color.
```

NOTES

If the starting point is already the color of color source you name (or any non-background when mode 1 is used), there is no change.

BUGS

None

statement/POKE statement/POKE

NAME

POKE -- Writes a value into a RAM memory

ABBREVIATION

p <shift> O

SYNOPSIS

POKE <address>,<value>

FUNCTION

The POKE command allows you to change any value in the computer RAM memory, and lets you modify many of the computer input/output registers. POKE is always followed by two numbers (or equations). The first number (<address>) is a location inside your computer's memory. This could have any value from 0 to 65535. The second number (<value>) is a value from 0 to 255, which is placed in the location, replacing any value that was there previously. This command can be used to control anything on the screen, from placing a character at that location to changing the color there.

INPUTS

<address> - memory address/location (0-65535)
<value>   - value to be stored in a given address (0-255)

RESULT

Given value is stored in a given memory location.

EXAMPLES

```
10 POKE 16000,8
    Sets location 16000 to 8.

20 POKE 16*1000,27
    Sets location 16000 to 27.
```

NOTES

None

BUGS

None

statement/PRINT statement/PRINT

NAME

PRINT -- Writes data to the screen

ABBREVIATION

?

SYNOPSIS

PRINT <printlist>

FUNCTION

The PRINT statement is the major output statement in BASIC. While the PRINT statement is the first BASIC statement most people learn to use, there are many subtleties to be mastered here as well. The word PRINT can be followed by any combinations of these items, which is considered the printlist (<printlist>):

Characters inside of quotes	"text lines"
Variable names	A B A$ X$
Functions	SIN(23) ABS(33)
Punctuation marks	; ,

The characters inside of quotes are often called literals because they are printed exactly as they appear. Variable names have the value they contain (either a number or a string) printed. Functions also have their number values printed. Punctuation marks are used to help format the data neatly on the screen. The comma (,) devides the screen into four columns for data, while the semicolon (;) doesn't add any spaces. Either mark can be used as the last symbol in the statement. This results in the next PRINT statement acting as if it is continuing the last PRINT statement.

INPUTS

<printlist> - items to be printed

RESULT

Given printlist is displayed on the screen.

EXAMPLES

```
10 PRINT "HELLO"
```

```
    20 A$="THERE":PRINT "HELLO,"A$
    30 A=4:B=2:PRINT A+B
    50 J=41:PRINT J;:PRINT J-1
    60 C=A+B:D=A-B:PRINT A;B;C,D
        Result:
        HELLO
        HELLO, THERE
         6
         41 40
         4 2 6    2
```

NOTES
    None

BUGS
    None

statement/PRINT   statement/PRINT

NAME
    PRINT USING -- Formats and writes data to the screen, file or device

ABBREVIATION
    ?us <shift>I

SYNOPSIS
    PRINT[<file>,]USING <formatlist>;<printlist>

FUNCTION
    These statements let you define the format of string and numeric items you want to print to the screen, printer, or another device. Put the format you want in quotes. This is the format list (<formatlist>). Then add a semicolon (;) and a list of what you want printed in the format for the print list (<printlist>). The list can be variables or the actual values you want printed.

Character		Numeric	String
Hash Sign	(#)	X	X
Plus	(+)	X	-
Minus	(-)	X	-
Decimal Point	(.)	X	-
Comma	(,)	X	-
Dollar Sign	($)	X	-
Four Carets	(^^^^)	X	-
Equal Sign	(=)	-	X
Greather Than Sign	(>)	-	X

The hash sign (#) reserves room for a single character in the output field. If the data item contains more characters than you have # in your format field, PRINT USING prints nothing. For a numeric item, the entire field is filled with asterisks (*). No numbers are printed.
For a STRING item, the string data is truncated at the bounds of the field. Only as many characters are printed as there are hash signs (#) in the format item. Truncation occurs on the right.
The plus (+) and minus (-) signs can be used in either the first or last position of a format field but not both. The plus sign is printed if the number is positive. The minus sign is printed if the number is negative.
If you use minus sign and the number is positive, a blank is printed in the character position indicated by the minus sign.
If you don't use either a plus or minus sign in your format field for a numeric data item, a minus sign is printed before the first digit or dollar symbol if the number is negative and no sign is printed if the number is positive. This means that you can print one character more if the number is positive. If there are too many digits to fit into the field specified by the # and + or - signs, then an overflow occurs and the field is filled with asterisks (*).
A decimal point (.) symbol designates the position of the decimal point in the number. You can only have one decimal point in any format field. If you don't specify a decimal point in your format field, the value is rounded to the nearest integer and printed without any decimal places. When you specify a decimal point, the number of digits preceding the decimal point (including the minus sign, if the value is negative) must not exceed the number of # before the decimal point. If there are too many digits an overflow occurs and the field is filled with asterisks

(*).

A comma (,) lets you place commas in numeric fields. The position of the comma in the format list indicates where the comma appears in a printed number. Only commas within a number are printed. Unused commas to the left of the first digit appear as the filler character. At least one # must precede the first comma in a field.

If you specify commas in a field and the number is negative, then a minus sign is printed as the first character even if the character position is specified as a comma.

A dollar sign ($) symbol shows that a dollar sign will be printed in the number. If you want the dollar sign to float (always be placed before the number), you must specify at least one # before the dollar sign. If you specify a dollar sign without a leading #, the dollar sign is printed in the position shown in the format field.

If you specify commas and/or a plus or minus sign in a format field with a dollar sign, your program prints a comma or sign before the dollar sign.

The four up arrows or carets (^^^^) symbol is used to specify that the number is to be printed in E+ format. You must use # in addition to the ^^^^ to specify the field width. The ^^^^ must appear after the # in the format field.

You must specify four carets (^^^^) when you want to print a number in E-format (scientific notation). If you specify more than one but fewer than four carets, you get a syntax error. If you specify more than four carets only the first four are used. The fifth caret (and subsequent carets) are interpreted literally as no text symbols.

An equal sign (=) is used to centre a string in the field. You specify the field width by the number of characters (# and =) in the format field. If the string contains fewer characters than the field width, the string is centered in the field. The right-most characters are truncated and the string fills the entire field.

A greater than sign (>) is used to right justify a string in a field. You specify the field width by the number of characters (# and =) in the format field. If the string contains fewer characters than the field width, the string is right justified in the field. If the string contains more characters than can be fit into the field, the right-most characters are truncated and the string fills the entire field.

INPUTS

<file> - logical number of target file/device
<formatlist> - printlist is formatted by using these format instructions
<printlist> - items to be printed

RESULT

Given printlist is formatted and displayed on the screen or written into a file or device.

EXAMPLES

```
5 X=32:Y=100.23:A$="CAT"
10 PRINT USING "$##.##";13.25,X,Y
20 PRINT USING "###>#";"CBM",A$
    When you RUN this, line 10 prints out:

    $13.25$32.00$*****

    PRINT USING prints ***** instead of Y value because Y has 5 digits,
    which does not conform to format list.
    Line 20 prints this:

       CBM CAT

    PRINT USING leaves three spaces before printing "CBM" as defined in
    format list.

10 PRINT USING "####";X
    For these values for X, this format displays:

    A = 12.34     12
    A = 567.89    568
    A = 123456    ****

10 PRINT USING "##.#+";-.01
    Result:

    0.01-

    Leading zero added.

10 PRINT USING "##.#-";1
    Result:
```

1.0

Trailing zero added.

```
10 PRINT USING "####";-100.5
```
Result:

-101

Rounded to no decimal places.

```
10 PRINT USING "####";-1000
```
Result:

****

Overflow because four digits and minus sign cannot fit in field.

```
10 PRINT USING "###.";10
```
Result:

10.

Decimal point added.

```
10 PRINT USING "#$##";1
```
Result:

$1

Leading $ sign.

NOTES
None

BUGS
None

statement/PRINT# statement/PRINT#

NAME
PRINT# -- Writes data to a file or a device

ABBREVIATION
p <shift> R

SYNOPSIS
```
PRINT#<file>,<printlist>
```

FUNCTION
There are a few differences between this statement and the PRINT. First of all, the word PRINT# is followed by a number, which refers to the device or data file previously OPENed. The number is followed by a comma, and a list of things to be PRINTed. The semicolon acts in the same manner for spacing as it does in the PRINT statement. The comma will send 10 spaces to most printers and can be used as a separator for disk files.

INPUTS
```
<file>      - logical number of target file/device
<printlist> - items to be printed
```

RESULT
Writes given data (<printlist>) to the target file/device.

EXAMPLES
```
100 PRINT#1,"HELLO THERE!",A$,B$,
```

NOTES
Some devices may not work with TAB and SPC.

BUGS
None

statement/PUDEF statement/PUDEF

NAME
PUDEF -- Redefines PRINT USING symbols

ABBREVIATION
p <shift> U

SYNOPSIS
PUDEF "<definition>"

FUNCTION
PUDEF lets you redefine up to 4 symbols in the PRINT USING statement. You can change blanks, commas, decimals points, and dollar signs into some other character by placing the new character in the correct position in the PUDEF control string.
Position 1 is the filler character. The default is a blank. Place a new character here when you want another chacter to appear in place of blanks.
Position 2 is the comma character. Default is a comma.
Position 3 is the decimal point.
Position 4 is the dollar sign.

INPUTS
<definition> - definition string for symbols (from left to right):
the first character defines a filler character,
the second character defines a comma,
the third character defines a decimal point, and
the fourth character defines a dollar sign

RESULT
Redefines four PRINT USIGN symbols: a filler character, a comma, a decimal point, and a dollar sign.

EXAMPLES
```
10 PUDEF "*"
    Prints * in the place of blanks.

20 PUDEF " &"
    Prints & in the place of commas.

30 PUDEF " .,"
    Prints decimal points in the place of commas, and commas in the
    place of decimal points.

40 PUDEF " .,£"
    Prints English pound sign in the place of $, decimal points in the
    place of commas, and commas in place of decimal points.
```

NOTES
None

BUGS
None

statement/READ statement/READ

NAME
READ -- Get information from DATA statements

ABBREVIATION
r <shift> E

SYNOPSIS
READ <variable>[,<...>,<variable>]

FUNCTION
This statement is used to get information from DATA statements into variables, where the data can be used. The READ statement variable list may contain both strings and numbers. Care must be taken to avoid reading strings where the READ statement expects a number, which produces an ERROR message.

INPUTS
<variable> - read data will be stored in this variable

RESULT
Data read from the DATA statements is stored in the target variables (<variable>).

EXAMPLES
    10 READ A$,G$,Y

NOTES
    None

BUGS
    None

statement/REM   statement/REM

NAME
    REM -- Attaches a note to the source code

ABBREVIATION
    None

SYNOPSIS
    REM [<message>]

FUNCTION
    The REMark is just a note to whoever is reading a LIST of the program. It may explain a section of the program, give information about the author, etc. REM statements in no way effect the operation of the program, except to add to its length (and therefore slow it down). The word REM may be followed by any text, although use of graphic characters gives strange results.

INPUTS
    <message> - any text

RESULT
    Attach a note to the source code so it can be read from the program listing.

EXAMPLES
    10 NEXT X:REM THIS LINE IS UNNECESSARY

NOTES
    None

BUGS
    None

statement/RESTORE   statement/RESTORE

NAME
    RESTORE -- Sets a DATA pointer

ABBREVIATION
    re <shift> S

SYNOPSIS
    RESTORE [<line>]

FUNCTION
    When executed in a program, the pointer to the item in a DATA statement which is to be read next is reset to the first item in the list. This gives you the ability to re-READ the information. If a line number (<line>) follows the RESTORE statement, the pointer is set to that line. Otherwise the pointer is reset to the first DATA statement in the program.

INPUTS
    <line> - a BASIC line number where DATA pointer should be set

RESULT
    Sets DATA pointer to the first item in a DATA item list or to the given BASIC line number which contains a DATA statement.

EXAMPLES
    10 RESTORE 200

NOTES
    None

BUGS
None

statement/RESUME statement/RESUME

NAME
RESUME -- Continues program execution after an error

ABBREVIATION
res <shift> U

SYNOPSIS
RESUME [<line>|NEXT]

FUNCTION
Used to return to execution after TRAPing an error. With no arguments, RESUME attempts to re-execute the line in which the error occured. RESUME NEXT resumes execution at the next statement following the statement containing the error; RESUME <line> will GOTO the specific line and begin execution there.

INPUTS
<line> - a BASIC line number where program execution should continue
NEXT - resume execution at the next statement

RESULT
Resumes execution after an error at the line where the error occured, or at the next BASIC line, or at a given BASIC line.

EXAMPLES
None

NOTES
None

BUGS
None

statement/RETURN statement/RETURN

NAME
RETURN -- Returns from a subroutine

ABBREVIATION
re <shift> T

SYNOPSIS
RETURN

FUNCTION
This statement is always used with the GOSUB statement. When the program encounters a RETURN statement, it goes to the statement immediately following the last GOSUB command executed. If no GOSUB was previously issued, then a RETURN WITHOUT GOSUB ERROR message is delivered, and program execution is stopped.

INPUTS
None

RESULT
Returns from a subroutine to the statement following the last subroutine call (GOSUB statement).

EXAMPLES
None

NOTES
None

BUGS
None

statement/SCALE statement/SCALE

NAME

SCALE -- Controls bit maps scaling

ABBREVIATION

sc <shift> A

SYNOPSIS

SCALE <scaling_flag>

FUNCTION

The scaling of the bit maps in multicolor and high resolution modes can be changed with the SCALE command. Entering:

SCALE 1

turns scaling on. Coordinates may then be scaled from 0 to 1023 in both x and y rather than the normal scale values, which are:

```
multicolor mode............ x = 0 to 159, y = 0 to 199
high resolution mode....... x = 0 to 319, y = 0 to 199
```

SCALE 0

turns scaling off.

INPUTS

<scaling_flag> - scaling mode: 0=no scaling, 1=scale

RESULT

Turns scaling on or off.

EXAMPLES

SCALE 1

NOTES

None

BUGS

None

None

statement/SCNCLR statement/SCNCLR

NAME

SCNCLR -- Clears the screen

ABBREVIATION

s <shift> C

SYNOPSIS

SCNCLR

FUNCTION

Clears the current screen, whether graphics, text, or both (split screen).

INPUTS

None

RESULT

Clears the current screen.

EXAMPLES

SCNCLR

NOTES

None

BUGS

None

statement/SOUND statement/SOUND

NAME

SOUND -- Produces a sound

ABBREVIATION

s <shift> O

SYNOPSIS

SOUND <voice>,<frq_control>,<duration>

FUNCTION

This statement produces a SOUND using one of the three voices (<voice>) with a frequency control (<frq_control>) in the range of 0-1023 for a duration (<duration>) of 0-65535 60ths of a second.
The Commodore 16 have 2 independent sound channels and ability to produce square and white noise timbres as follows:

Voice	Sound channel	Wave shape
1	1	square
2	2	square
3	2	noise

If a SOUND for voice n is requested, and the previous SOUND for the same n is still playing, BASIC waits for the previous SOUND to complete.
SOUND with a duration of 0 is a special case. It causes BASIC to turn off the current SOUND for that voice immediately, regardless of the time remaining on the previous SOUND.

INPUTS

<voice>   - voice number (1-3)
<frq_control> - sound register value (0-1023)
<duration>   - duration of a sound in 60ths of a second (0-65535)

RESULT

Cuts off or produces a sound with square or white noise timbre.

EXAMPLES

```
SOUND 2,800,3600
```

Plays a note using voice 2 with frequency set at 800 for one minute.

NOTES

The sound register value (<frq_control>) does not correspond directly to the real sound frequence. If you want to produce a sound with a certain frequency use the following formulas to find the sound register value for the desired frequency:

formula for computers using PAL television standard

```
SOUND REGISTER VALUE = 1024-(111840.45/FREQUENCY)
```

formula for computers using NTSC television standard

```
SOUND REGISTER VALUE = 1024-(111860.781/FREQUENCY)
```

BUGS

None

statement/SSHAPE   statement/SSHAPE

NAME

SSHAPE -- Saves a rectangular graphic area into a string variable

ABBREVIATION

s <shift> S

SYNOPSIS

SSHAPE <shape>,<left>,<top>[,<right>,<bottom>]

FUNCTION

This statement is used to save a rectangular area of multicolor or high resolution screen using BASIC string variable.
Because BASIC limits string lengths to 255 characters, the size of the area you may save is limited. The string size required can be calculated using one of the following (unscaled) formulas:

```
L(mcm) = INT((ABS(<left>-<right>)+1)/4+.99)*(ABS(<top>-<bottom>)+1)+
L(h-r) = INT((ABS(<left>-<right>)+1)/8+.99)*(ABS(<top>-<bottom>)+1)+
```

(mcm) refers to multi-color mode; (h-r) is high resolution mode.

The shape is saved row by row. The last four bytes of the string contain the column and row lengths less one (i.e.: ABS(<left>-<right>)) in low/high byte format (if scaled divide the lengths by 3.2 (X) and 5.12 (Y)).

INPUTS

<shape> - string variable where shape sould be stored
<left> - scaled corner coordinate
<top> - scaled corner coordinate
<right> - scaled corner coordinate
<bottom> - scaled corner coordinate

RESULT

Saves a defined rectangular graphic area into a BASIC string variable.

EXAMPLES

SSHAPE V$,0,0

Saves screen area from the upper left corner to where the cursor is positioned under the name V$.

NOTES

None

BUGS

None

statement/STOP statement/STOP

NAME

STOP -- Halts the program execution

ABBREVIATION

s <shift> T

SYNOPSIS

STOP

FUNCTION

This statement halts the program. A message, BREAK IN LINE <line>, where the <line> is the line number containing the STOP. The program can be re-started at the statement following STOP if you use the CONT command. The STOP statement is usually used while debugging a program.

INPUTS

None

RESULT

Halts the program execution.

EXAMPLES

100 STOP

NOTES

None

BUGS

None

statement/SYS statement/SYS

NAME

SYS -- Executes a machine language program

ABBREVIATION

s <shift> Y

SYNOPSIS

SYS <address>

FUNCTION

The word SYS is followed by a decimal number or numeric variable in the range 0 to 65535. The program begins executing the machine language program starting at that memory location. (This is similar to the USR function, but does not pass a parameter.) However, parameters can be passed anyway using the following memory locations:

2034 = Accumulator
2035 = X register
2036 = Y register

These can be used both before SYS to set the processor registers (for example POKE

2034,255) and after the return from the machine language routine to check the results (for example A=PEEK(2034))

(For further communication, you can of course also poke and peek to memory locations that you design your ML routine to use.)

INPUTS

<address> - memory address (0-65535)

RESULT

Begins executing a machine language program from the given memory location.

EXAMPLES

None

NOTES

None

BUGS

None

statement/TRAP statement/TRAP

NAME

TRAP -- Turns on or off error interception

ABBREVIATION

t <shift> R

SYNOPSIS

TRAP [<line>]

FUNCTION

When turned on, TRAP intercepts all error conditions (including the <run/stop> key) exept "UNDEF'D STATEMENT ERROR". In the event of any execution error, the error flag is set, and execution is transferred to the line number named in the TRAP statement (<line>). The line number in which the error occured can be found by using the system variable EL. The string function ERR$(ER) gives the error message corresponding to any error condition ER.
TRAP with no line number argument turns off error TRAPping.

INPUTS

<line> - BASIC line number where program execution should continue when an error occures

RESULT

When line number has been given turns on error interception, otherwise turns it off.

EXAMPLES

```
200 TRAP 210
210 PRINT "AN ERROR OCCURED IN LINE"EL".":STOP
```

NOTES

An error in a TRAP routine cannot be trapped. The RESUME statement can be used to resume execution.

BUGS

None

statement/TROFF statement/TROFF

NAME

TROFF -- Turns trace mode off

ABBREVIATION

tro <shift> F

SYNOPSIS

TROFF

FUNCTION

This statement turns trace mode off.

INPUTS
    None

RESULT
    Exits trace mode.

EXAMPLES
    None

NOTES
    None

BUGS
    None

statement/TRON   statement/TRON

NAME
    TRON -- Turns trace mode on

ABBREVIATION
    tr <shift> O

SYNOPSIS
    TRON

FUNCTION
    TRON is used in program debugging. This statement begins trace mode.
    When you are in trace mode, as each statement executes, the line number
    of that statement is printed.

INPUTS
    None

RESULT
    Begins the trace mode.

EXAMPLES
    None

NOTES
    None

BUGS
    None

statement/VOL   statement/VOL

NAME
    VOL -- Sets sound volume level

ABBREVIATION
    v <shift> O

SYNOPSIS
    VOL <volume>

FUNCTION
    Sets the current VOLume level for SOUND commands. VOLume may be set from
    0 to, where 8 is maximum volume, and 0 is off. VOL affects all channels.

INPUTS
    <volume> - sound volume (0-8)

RESULT
    Sets sound volume level.

EXAMPLES
```
10 VOL 8
```
        Sets sound volume level to the maximum.

NOTES
    None

BUGS

None

statement/WAIT statement/WAIT

NAME
    WAIT -- Waits for a change of memory address

ABBREVIATION
    w <shift> A

SYNOPSIS
    WAIT <address>,<ctrl_value1>[,<ctrl_value2>]

FUNCTION
    The WAIT statement is used to halt the program until the contents of a location in memory changes in a specific way. The address (<address>) must be in the range from 0 to 65535. Value 1 (<ctrl_value1>) and value 2 (<ctrl_value2>) must be in the range from 0 to 255.
    The content of the memory location is first exclusive-ORed (XOR) with value 2 (if present), and then logically ANDed (AND) with value 1. If the result is zero, the program checks the memory location again. When the result is not zero, the program continues with the next statement.

INPUTS
    <address>   - memory location to be monitored (0-65535)
    <ctrl_value1> - first control value (0-255)
    <ctrl_value2> - second control value (0-255)

RESULT
    Halts program execution until the contents of a given memory address changes.

EXAMPLES
    None

NOTES
    None

BUGS
    None

VARIABLES

Your computer uses three types of variables in BASIC. These are: normal numeric, integer numeric, and string (alphanumeric) variables.

NUMERIC VARIABLES

Normal numeric variables, also called floating point variables, can have any value from ^ -38 to ^ +38, with up to nine digits of accuracy. When a number becomes larger than nine digits can show, as in 10 ^ -10 or 10 ^ +10, your computer displays it in scientific notation form, with the number normalized to 1 digit and eight decimal places, followed by the letter E and the power of ten by which the number is multiplied. For example, the number 12345678901 is displayed as 1.23456789E+10.

INTEGER VARIABLES

Integer variables can be used when the number is from +32767 to -32768, and with no fractional portion. An integer variable is a number like 5, 10, or -100. Integers take up less space than floating point variables when used in an array.

STRING VARIABLES

String variables are those used for character data, which may contain numbers, letters, and any other character that your computer can make. An example of a string variable is "COMMODORE 16".

VARIABLE NAMES

Variable names may consist of a single letter, a letter followed by a number, or two letters. Variable names may be longer than 2 characters, but only the first two are significant.
An integer variable is specified by using the percent (%) sign after the variable name. String variables have the dollar sign ($) after their names.

Examples:

Numeric Variable Names: A, A5, BZ
Integer Variable Names: A%, A5%, BZ%
String Variable Names : A$, A5$,BZ$

ARRAYS

Arrays are lists of variables with the same name, using an extra number (or numbers) to specify an element of the array. Arrays are defined using the DIM statement, and may be floating point, integer, or string variables arrays. The array variable name is followed by a set of parentheses ( ) enclosing the number of the variable in the list.

Examples:

A(7), BZ%(11), A$(87)

Arrays may have more than one dimension. A two dimensional array may be viewed as having rows and columns, with the first number identifying the column and the second number in the parentheses identifying the row (as if specifying a certain grid on a map).

Examples:

A(7,2), BZ%(2,3,4), A$(3,2)

RESERVED VARIABLE NAMES

There are seven variable names which are reserved for use by the computer, and may not be used for another purpose. These are the variables:

variable/DS variable/DS

NAME
DS -- Disk drive's status

DESCRIPTION
The variable DS reads the disk drive command channel, and returns the current status of the drive.
DS is used after a disk operation (like DLOAD or DSAVE) to find out why the red error light on the disk drive is blinking.

EXAMPLES
None

NOTES
None

variable/DS$ variable/DS$

NAME
DS$ -- Disk drive's status in words

DESCRIPTION
The variable DS$ reads the disk drive command channel, and returns the current status of the drive in words.
DS$ is used after a disk operation (like DLOAD or DSAVE) to find out why the red error light on the disk drive is blinking.

EXAMPLES
None

NOTES

None

variable/EL variable/EL

NAME
    EL -- Last error line

DESCRIPTION
    The variable EL is used typically in error trapping routines. EL stores the line number where last error occured.

EXAMPLES
    None

NOTES
    None

variable/ER variable/ER

NAME
    ER -- Last error line

DESCRIPTION
    The variable ER is used typically in error trapping routines. ER stores the last error (error condition number) encountered since the program was run.

EXAMPLES
    None

NOTES
    None

variable/ST variable/ST

NAME
    ST -- Input/output status

DESCRIPTION
    ST is a status variable for input and output (exept normal screen/keyboard operations). The value of ST depends on the results of the last input/output operation.

EXAMPLES
    None

NOTES
    None

    None

variable/TI variable/TI

NAME
    TI -- Clock value

DESCRIPTION
    TI variable contains the current value of the clock in 1/60ths of a second.

EXAMPLES
    None

NOTES
    None

variable/TI$ variable/TI$

NAME
    TI$ -- Current time

DESCRIPTION
    TI$ is a string that reads the value of the real-time clock as a 24 hour clock. The first two characters of TI$ contain the hour, the 3rd and 4th characters are the minutes, and the 5th and 6th characters are the seconds. This variable can be set to any value (so long as all characters are numbers), and will be automatically updated as a 24 hour clock.
    The value of the clock is lost when computer is turned off. It starts at zero when computer is turned on, and is reset to zero when the value of the clock exeeds 235959 (23 hours, 59 minutes and 59 seconds).

EXAMPLES
    TI$ = "101530"
        Sets the clock to 10:15 and 30 seconds (AM).

NOTES
    None

BASIC ERROR MESSAGES

These error messages are printed by BASIC. You can also PRINT the messages through the use of the ERR$ function. The error number refers only to the number assigned to the error for use with this function.

basic_error/01_TOO_MANY_FILES basic_error/01_TOO_MANY_FILES

NUMBER
    1

MESSAGE
    TOO MANY FILES

DESCRIPTION
    There is a limit of 10 files OPEN at one time.

basic_error/02_FILE_OPEN basic_error/02_FILE_OPEN

NUMBER
    2

MESSAGE
    FILE OPEN

DESCRIPTION
    An attempt was made to open a file using the number of an already open file.

basic_error/03_FILE_NOT_OPEN basic_error/03_FILE_NOT_OPEN

NUMBER
    3

MESSAGE
    FILE NOT OPEN

DESCRIPTION
    The file number specified in an I/O statement must be opened before use.

basic_error/04_FILE_NOT_FOUND basic_error/04_FILE_NOT_FOUND

NUMBER
    4

MESSAGE
    FILE NOT FOUND

DESCRIPTION
    No file with that name exists (disk).

basic_error/05_DEVICE_NOT_PRESENT basic_error/05_DEVICE_NOT_PRESENT

NUMBER

5

MESSAGE
    DEVICE NOT PRESENT

DESCRIPTION
    The required I/O device not available.

basic_error/06_NOT_INPUT_FILE basic_error/06_NOT_INPUT_FILE

NUMBER
    6

MESSAGE
    NOT INPUT FILE

DESCRIPTION
    An attempt made to GET or INPUT data from a file that was specified as output only.

basic_error/07_NOT_OUTPUT_FILE basic_error/07_NOT_OUTPUT_FILE

NUMBER
    7

MESSAGE
    NOT OUTPUT FILE

DESCRIPTION
    An attempt made to send data to a file that was specified as input only.

basic_error/08_MISSING_FILE_NAME basic_error/08_MISSING_FILE_NAME

NUMBER
    8

MESSAGE
    MISSING FILE NAME

DESCRIPTION
    An OPEN, LOAD, or SAVE to the disk generally requires a file name.

basic_error/09_ILLEGAL_DEVICE_NUMBER basic_error/09_ILLEGAL_DEVICE_NUMBER

NUMBER
    9

MESSAGE
    ILLEGAL DEVICE NUMBER

DESCRIPTION
    An attempt made to use a device improperly (SAVE to the screen, etc.).

basic_error/10_NEXT_WITHOUT_FOR basic_error/10_NEXT_WITHOUT_FOR

NUMBER
    10

MESSAGE
    NEXT WITHOUT FOR

DESCRIPTION
    Either loops are nested incorrectly, or there is a variable name in a NEXT statement that does not correspond with one in a FOR.

basic_error/11_SYNTAX_ERROR basic_error/11_SYNTAX_ERROR

NUMBER
    11

MESSAGE
    SYNTAX ERROR

DESCRIPTION
    A statement is unrecognizable by BASIC. This could be because of missing or extra parenthesis, misspelled keyword, etc.

basic_error/12_RETURN_WITHOUT_GOSUB basic_error/12_RETURN_WITHOUT_GOSUB

NUMBER

12

MESSAGE
RETURN WITHOUT GOSUB

DESCRIPTION
A RETURN statement encountered when no GOSUB statement was active.

basic_error/13_OUT_OF_DATA basic_error/13_OUT_OF_DATA

NUMBER
13

MESSAGE
OUT OF DATA

DESCRIPTION
A READ statement encountered, without data left unREAD.

basic_error/14_ILLEGAL_QUANTITY basic_error/14_ILLEGAL_QUANTITY

NUMBER
14

MESSAGE
ILLEGAL QUANTITY

DESCRIPTION
A number used as the argument of a function or statement is outside the allowable range.

basic_error/15_OVERFLOW basic_error/15_OVERFLOW

NUMBER
15

MESSAGE
15 OVERFLOW

DESCRIPTION
The result of a computation is larger than the largest number allowed (1.701411833E+38).

basic_error/16_OUT_OF_MEMORY basic_error/16_OUT_OF_MEMORY

NUMBER
16

MESSAGE
OUT OF MEMORY

DESCRIPTION
Either there is no more room for program and program variables, or there are too many DO, FOR, or GOSUB statements in effect.

basic_error/17_UNDEF'D_STATEMENT basic_error/17_UNDEF'D_STATEMENT

NUMBER
17

MESSAGE
UNDEF'D STATEMENT

DESCRIPTION
A line number referenced does not exist in the program.

basic_error/18_BAD_SUBSCRIPT basic_error/18_BAD_SUBSCRIPT

NUMBER
18

MESSAGE
BAD SUBSCRIPT

DESCRIPTION
The program tried to reference an element of an array out of the range specified by the DIM statement.

basic_error/19_REDIM'D_ARRAY basic_error/19_REDIM'D_ARRAY

NUMBER
    19

MESSAGE
    REDIM'D ARRAY

DESCRIPTION
    An array can only be DIMensioned once. If an array is referenced before that array is DIM'd, an automatic DIM (to 10) is performed.

basic_error/20_DIVISION_BY_ZERO   basic_error/20_DIVISION_BY_ZERO

NUMBER
    20

MESSAGE
    DIVISION BY ZERO

DESCRIPTION
    Division by zero is not allowed.

basic_error/21_ILLEGAL_DIRECT   basic_error/21_ILLEGAL_DIRECT

NUMBER
    21

MESSAGE
    ILLEGAL DIRECT

DESCRIPTION
    INPUT or GET statements are only allowed within a program.

basic_error/22_TYPE_MISMATCH   basic_error/22_TYPE_MISMATCH

NUMBER
    22

MESSAGE
    TYPE MISMATCH

DESCRIPTION
    This occurs when a number is used in place of a string or vice-versa.

basic_error/23_STRING_TOO_LONG   basic_error/23_STRING_TOO_LONG

NUMBER
    23

MESSAGE
    STRING TOO LONG

DESCRIPTION
    A string can contain up to 255 characters.

basic_error/24_FILE_DATA   basic_error/24_FILE_DATA

NUMBER
    24

MESSAGE
    FILE DATA

DESCRIPTION
    Bad data read from a tape.

basic_error/25_FORMULA_TOO_COMPLEX   basic_error/25_FORMULA_TOO_COMPLEX

NUMBER
    25

MESSAGE
    FORMULA TOO COMPLEX

DESCRIPTION
    Simplify the expression (break into two parts or use fewer parentheses).

basic_error/26_CAN'T_CONTINUE   basic_error/26_CAN'T_CONTINUE

NUMBER
    26

MESSAGE
    CAN'T CONTINUE

DESCRIPTION
    The CONT command does not work if the program was not RUN, there was an error, or a line has been edited.

basic_error/27_UNDEF'D_FUNCTION basic_error/27_UNDEF'D_FUNCTION

NUMBER
    27

MESSAGE
    UNDEF'D FUNCTION

DESCRIPTION
    A user defined function referenced that was never defined.

basic_error/28_VERIFY basic_error/28_VERIFY

NUMBER
    28

MESSAGE
    VERIFY

DESCRIPTION
    The program on tape or disk does not match the program in memory.

basic_error/29_LOAD basic_error/29_LOAD

NUMBER
    29

MESSAGE
    LOAD

DESCRIPTION
    There was a problem loading. Try again.

basic_error/30_BREAK basic_error/30_BREAK

NUMBER
    30

MESSAGE
    BREAK

DESCRIPTION
    The stop key was hit to halt program execution.

basic_error/31_CAN'T_RESUME basic_error/31_CAN'T_RESUME

NUMBER
    31

MESSAGE
    CAN'T RESUME

DESCRIPTION
    A RESUME statement encountered without TRAP statement in effect.

basic_error/32_LOOP_NOT_FOUND basic_error/32_LOOP_NOT_FOUND

NUMBER
    32

MESSAGE
    LOOP NOT FOUND

DESCRIPTION
    The program has encountered a DO statement and cannot find the corresponding LOOP.

basic_error/33_LOOP_WITHOUT_DO basic_error/33_LOOP_WITHOUT_DO

NUMBER
    33

MESSAGE
    LOOP WITHOUT DO

DESCRIPTION
    LOOP encountered without a DO statement active.

basic_error/34_DIRECT_MODE_ONLY basic_error/34_DIRECT_MODE_ONLY

NUMBER
    34

MESSAGE
    DIRECT MODE ONLY

DESCRIPTION
    This command is allowed only in direct mode, not from a program.

basic_error/35_NO_GRAPHICS_AREA basic_error/35_NO_GRAPHICS_AREA

NUMBER
    35

MESSAGE
    NO GRAPHICS AREA

DESCRIPTION
    A command (DRAW, BOX, etc.) to create graphics encountered before the GRAPHIC command was executed.

basic_error/36_BAD_DISK basic_error/36_BAD_DISK

NUMBER
    36

MESSAGE
    BAD DISK

DESCRIPTION
    An attempt failed to HEADER a disk, because the quick header method (no ID) was attempted on an unformatted disk, or the disk is bad.

DISK ERROR MESSAGES

NOTES

    Error message numbers 02-19, 35-38, 40-49, 53-59, and 68-69 should be ignored. Message number 01 (<deleted>) gives information about the number of files deleted with the SCRATCH command.

disk_error/20_READ_ERROR disk_error/20_READ_ERROR

NUMBER
    20

MESSAGE
    READ ERROR

DESCRIPTION
    Block header not found.

    The disk controller is unable to locate the header of the requested data block. Caused by an illegal sector number, or the header has been destroyed.

disk_error/21_READ_ERROR disk_error/21_READ_ERROR

NUMBER
    21

MESSAGE
    READ ERROR

DESCRIPTION
    No sync character.

The disk controller is unable to detect a sync mark on the desired track. Caused by misalignment of the read/write head, no disk is present, or unformatted or improperly seated disk. Can also indicate a hardware failure.

disk_error/22_READ_ERROR disk_error/22_READ_ERROR

NUMBER
22

MESSAGE
READ ERROR

DESCRIPTION
Data block not present.

The disk controller has been requested to read or verify a data block that was not properly written. This error message occurs in conjunction with the BLOCK commands and indicates an illegal track and/or sector request.

disk_error/23_READ_ERROR disk_error/23_READ_ERROR

NUMBER
23

MESSAGE
READ ERROR

DESCRIPTION
Checksum error in data block.

This error message indicates that there is an error in one or more of the data types. The data has been read into the DOS memory, but the checksum over the data is in error. This message may also indicate grounding problems.

disk_error/24_READ_ERROR disk_error/24_READ_ERROR

NUMBER
24

MESSAGE
READ ERROR

DESCRIPTION
Byte decoding error.

The data or header has been read into the DOS memory, but a hardware error has been created due to an invalid bit pattern in the data byte. This message may also indicate grounding problems.

disk_error/25_WRITE_ERROR disk_error/25_WRITE_ERROR

NUMBER
25

MESSAGE
WRITE ERROR

DESCRIPTION
Write-verify error.

This message is generated if the controller detects a mismatch between the written data and the data in the DOS memory.

disk_error/26_WRITE_PROTECT_ON disk_error/26_WRITE_PROTECT_ON

NUMBER
26

MESSAGE
WRITE PROTECT ON

DESCRIPTION
This message is generated when the controller has been requested to write a data block while the write protect switch is depressed. Typically, this is caused by using a disk with a write protect tab over the notch.

disk_error/27_READ_ERROR disk_error/27_READ_ERROR

NUMBER
27

MESSAGE
READ ERROR

DESCRIPTION
Checksum error in header.

The controller has detected an error in the header of the requested data block. The block has not been read into the DOS memory. This message may also indicate grounding problems.

disk_error/28_WRITE_ERROR disk_error/28_WRITE_ERROR

NUMBER
28

MESSAGE
WRITE ERROR

DESCRIPTION
Too long data block.

The controller attempts to detect the sync mark of the next header after writing a data block. If the sync mark does not appear within a pre-determined time, the error message is generated. The error is caused by a bad disk format (the data extends into the next block), or by hardware failure.

disk_error/29_DISK_ID_MISMATCH disk_error/29_DISK_ID_MISMATCH

NUMBER
29

MESSAGE
DISK ID MISMATCH

DESCRIPTION
This message is generated when the controller has been requested to access a disk which has not been initialized. The message can also occur if a disk has a bad header.

disk_error/30_SYNTAX_ERROR disk_error/30_SYNTAX_ERROR

NUMBER
30

MESSAGE
SYNTAX ERROR

DESCRIPTION
Error in general syntax.

The DOS cannot interpret the command sent to the command channel. Typically, this is caused by an illegal number of file names, or patterns are illegally used. For example, two file names may appear on the left side of the COPY command.

disk_error/31_SYNTAX_ERROR disk_error/31_SYNTAX_ERROR

NUMBER
31

MESSAGE
SYNTAX ERROR

DESCRIPTION
Invalid command.

The DOS does not recognize the command. The command must start in the first position.

disk_error/32_SYNTAX_ERROR disk_error/32_SYNTAX_ERROR

NUMBER
32

MESSAGE
    SYNTAX ERROR

DESCRIPTION
    Invalid command.

    The command sent is longer than 58 characters.

disk_error/33_SYNTAX_ERROR   disk_error/33_SYNTAX_ERROR

NUMBER
    33

MESSAGE
    SYNTAX ERROR

DESCRIPTION
    Invalid file name.

    Pattern matching is invalidly used in the OPEN or SAVE command.

disk_error/34_SYNTAX_ERROR   disk_error/34_SYNTAX_ERROR

NUMBER
    34

MESSAGE
    SYNTAX ERROR

DESCRIPTION
    No file given.

    The file name was left out of a command or the DOS does not recognize it as such. Typically, a colon (:) has been left out of the command.

disk_error/39_SYNTAX_ERROR   disk_error/39_SYNTAX_ERROR

NUMBER
    39

MESSAGE
    SYNTAX ERROR

DESCRIPTION
    Invalid command.

    This error may result if the command sent to command channel (secondary address 15) is unrecognized by the DOS.

disk_error/50_RECORD_NOT_PRESENT   disk_error/50_RECORD_NOT_PRESENT

NUMBER
    50

MESSAGE
    RECORD NOT PRESENT

DESCRIPTION
    Result of disk reading past the last record through INPUT#, or GET# commands. This message will also occur after positioning to a record beyond end of file in a relative file. If the intent is to expand the file by adding the new record (with a PRINT# command), the error message may be ignored. INPUT or GET should not be attempted after this error is detected without first repositioning.

disk_error/51_OVERFLOW_IN_RECORD   disk_error/51_OVERFLOW_IN_RECORD

NUMBER
    51

MESSAGE
    OVERFLOW IN RECORD

DESCRIPTION
    PRINT# statement exceeds record boundary. Information is truncated. Since the carriage return which is sent as a record terminator is counted in the record size, this message will occur if the total characters in the record (including the final carriage return) exceeds the defined size.

disk_error/52_FILE_TOO_LARGE disk_error/52_FILE_TOO_LARGE

NUMBER
52

MESSAGE
FILE TOO LARGE

DESCRIPTION
Record position within a relative file indicates that disk overflow will result.

disk_error/60_WRITE_FILE_OPEN disk_error/60_WRITE_FILE_OPEN

NUMBER
60

MESSAGE
WRITE FILE OPEN

DESCRIPTION
This message is generated when a write file that has not been closed is being opened for reading.

disk_error/61_FILE_NOT_OPEN disk_error/61_FILE_NOT_OPEN

NUMBER
61

MESSAGE
FILE NOT OPEN

DESCRIPTION
This message is generated when a file is being accessed that has not been opened in the DOS. Sometimes, in this case, a message is not generated; the request simply ignored.

disk_error/62_FILE_NOT_FOUND disk_error/62_FILE_NOT_FOUND

NUMBER
62

MESSAGE
FILE NOT FOUND

DESCRIPTION
The requested file does not exist on the indicated drive.

disk_error/63_FILE_EXISTS disk_error/63_FILE_EXISTS

NUMBER
63

MESSAGE
FILE EXISTS

DESCRIPTION
The file name of the file being created already exists on the disk.

disk_error/64_FILE_TYPE_MISMATCH disk_error/64_FILE_TYPE_MISMATCH

NUMBER
64

MESSAGE
FILE TYPE MISMATCH

DESCRIPTION
The file type does not match the file type in the directory entry for the requested file.

disk_error/65_NO_BLOCK disk_error/65_NO_BLOCK

NUMBER
65

MESSAGE
NO BLOCK

DESCRIPTION

This message occurs in conjunction with the B-A command. It indicates that the block to be allocated has been previously allocated. The parameters indicate the track and sector available with the next highest number. If the parameters are zero (0), then all blocks higher in number are in use.

disk_error/66_ILLEGAL_TRACK_AND_SECTOR disk_error/66_ILLEGAL_TRACK_AND_SECTOR

NUMBER
66

MESSAGE
ILLEGAL TRACK AND SECTOR

DESCRIPTION
The DOS has attempted to access a track or block which does not exist in the format being used. This may indicate a problem reading the pointer to the next block.

disk_error/67_ILLEGAL_SYSTEM_T_OR_S disk_error/67_ILLEGAL_SYSTEM_T_OR_S

NUMBER
67

MESSAGE
ILLEGAL SYSTEM T OR S

DESCRIPTION
This special error message indicates an illegal system track or sector.

disk_error/70_NO_CHANNEL disk_error/70_NO_CHANNEL

NUMBER
70

MESSAGE
NO CHANNEL

DESCRIPTION
No channel available.

The requested channel is not available, or all channels are in use. A maximum of five sequential files may be opened at one time to the DOS. Direct access channels may have six opened files.

disk_error/71_DIRECTORY_ERROR disk_error/71_DIRECTORY_ERROR

NUMBER
71

MESSAGE
DIRECTORY ERROR

DESCRIPTION
The BAM (Block Availability Map) does not match the internal count. There is a problem in the BAM allocation or the BAM has been overwritten in DOS memory. To correct this problem reinitialize the disk to restore the BAM in memory. Some active files may be terminated by the corrective action.

disk_error/72_DISK_FULL disk_error/72_DISK_FULL

NUMBER
72

MESSAGE
DISK FULL

DESCRIPTION
Either the blocks on the disk are used or the directory is at its entry limit. DISK FULL is sent when two blocks are available on the 1541 to allow the current file to be closed.

disk_error/73_DOS_MISMATCH disk_error/73_DOS_MISMATCH

NUMBER
73

MESSAGE
DOS MISMATCH

DESCRIPTION

DOS 1 and 2 are read compatible but not write compatible. Disks may be interchangeably read with either DOS, but a disk formatted on one version cannot be written upon with the other version because the format is different. This error is displayed whenever an attempt is made to write upon a disk which has been formatted in a non-compatible format. (A utility routine is available to assist in converting from one format to another.) This message may also appear after power up.

NOTES

Error number 73 in CBM DOS V2.6 1541.

disk_error/74_DRIVE_NOT_READY disk_error/74_DRIVE_NOT_READY

NUMBER

74

MESSAGE

DRIVE NOT READY

DESCRIPTION

An attempt has been made to access the Floppy Disk Drive without any disk present.

BASIC ABBREVIATIONS

To obtain Basic keywords without having to type the whole command, press the letter(s) on the left and then shift key and the letter on the right. The shifted character appears as a graphics character but when the line is listed, the abbreviation will be expanded out into the full word. The abbreviation for a keyword is generally the first letter of the keyword and the second letter shifted, but this may vary.

Keyword	**First letter(s)**	**Shifted Letter**
ABS	A	R
ASC	A	S
ATN	A	T
AUTO	A	U
BACKUP	B	A
BOX	B	O
CHAR	CH	A
CHR$	C	H
CIRCLE	C	I
CLOSE	CL	O
CLR	C	L
CMD	C	M
COLLECT	COL	L
COLOR	CO	L
CONT	C	O
COPY	CO	P
COS	none	
DATA	D	A
DEC	none	
DEF FN	D	E
DELETE	DE	L
DIM	D	I
DIRECTORY	DI	R
DLOAD	D	L
DO	none	
DRAW	D	R
DSAVE	D	S
END	E	N
ERR$	E	R
EXP	E	X
FOR	F	O
FRE	F	R
GET	G	E
GET#	none	
GETKEY	GETK	E
GOSUB	GO	S

```
 GOTO        G                 O
 GRAPHIC     G                 R
 GSHAPE      G                 S
 HEADER      HE                A
-------------------------------------------------------------------------------
 HELP        HE                L
 HEX$        H                 E
 IF          none
 INPUT       none
 INPUT#      I                 N
 INSTR       IN                S
 INT         none
 JOY         J                 O
 KEY         K                 E
 LEFT$       LE                F
-------------------------------------------------------------------------------
 LEN         none
 LET         L                 E
 LIST        L                 I
 LOAD        L                 O
 LOCATE      LO                C
 LOG         none
 LOOP        LO                O
 MID$        M                 I
 MONITOR     M                 O
 NEW         none
-------------------------------------------------------------------------------
 NEXT        N                 E
 ON..GOSUB   ON..GO            S
 ON..GOTO    ON..G             O
 OPEN        O                 P
 PAINT       P                 A
 PEEK        P                 E
 POKE        P                 O
 POS         none
 PRINT       ?
 PRINT USING ?US               I
-------------------------------------------------------------------------------
 PRINT#      P                 R
 PUDEF       P                 U
 RCLR        R                 C
 RDOT        R                 D
 READ        R                 E
 REM         none
 RENAME      RE                N
 RENUMBER    REN               U
 RESTORE     RE                S
 RESUME      RES               U
-------------------------------------------------------------------------------
 RETURN      R                 T
 RGR         R                 G
 RIGHT$      R                 I
 RLUM        R                 L
 RND         R                 N
 RUN         R                 U
 SAVE        S                 A
 SCALE       SC                A
 SCNCLR      S                 C
 SCRATCH     SC                R
-------------------------------------------------------------------------------
 SGN         S                 G
 SIN         S                 I
 SOUND       S                 O
 SPC         S                 P
 SQR         S                 Q
 SSHAPE      S                 S
 STOP        S                 T
 STR$        ST                R
 SYS         S                 Y
 TAB(        T                 A
-------------------------------------------------------------------------------
 TAN         none
 TRAP        T                 R
 TROFF       TRO               F
 TRON        TR                O
 UNTIL       U                 N
 USR         U                 S
 VAL         none
 VERIFY      V                 E
 VOL         V                 O
```

WAIT	W	A
WHILE	W	H

PETASCII CODES

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0		16	SPACE	32	ø	48
	1	CRSR ↓	17	!	33	1	49
	2	RVS ON	18	"	34	2	50
Stop	3	HOME	19	#	35	3	51
	4	DEL	20	$	36	4	52
White	5		21	%	37	5	53
	6		22	&	38	6	54
	7		23	'	39	7	55
Disable Shift	8		24	(	40	8	56
Enable Shift	9		25	)	41	9	57
	10		26	*	42	:	58
	11	ESC	27	+	43	;	59
	12	Red	28	,	44	<	60
RETURN	13	CRSR ⇒	29	—	45	=	61
LoCase	14	Green	30	.	46	>	62
	15	Blue	31	/	47	?	63

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
@	64	U	85		106		127
A	65	V	86		107		128
B	66	W	87		108	Orange	129
C	67	X	88		109	Flash On	130
D	68	Y	89		110	Shift + run/stop	131
E	69	Z	90		111	Flash Off	132
F	70	[(Ä)	91		112	f1	133
G	71	£ (Ö)	92	●	113	f3	134
H	72	] (Å)	93		114	f5	135
I	73	↑	94	♥	115	f7	136
J	74	←	95		111	f2	137
K	75		96		117	f4	138
I	76	♠	97	╳	118	f6	139
M	77		98	○	119		140
N	78		99	♣	120	SHIFT RETURN	141
O	79		100		121	UCase	142
P	80		101	♦	122		143
Q	81		102	┼	123	Black	144
R	82		103		124	↑ CRSR	145
S	83		104		125	RVS OFF	146
T	84		105	π	126	CLR	147

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
INST	148	Cyan	159		170		181
Brown	149	SPACE	160		171		182
Lt. Green	150		161		172		183
Pink	151		162		173		184
Green	152		163		174		185
Lt. Blue	153		164		175		186
Dp. Pur	154		165		176		187
Lt. Green	155		166		177		188
Purple	156		167		178		189
CRSR	157		168		179		190
Yellow	158		169		180		191

Codes 192-223 are same as 96-127.
Codes 224-254 are same as 160-190.
Code 255 is the same as code 126.

Code 5 appears between quotes as code 69, but reversed.
Code 28 appear between quotes as code 92, but reversed.
Codes 30-31 appear between quotes as codes 94-95, but reversed.
Code 129 appears between quotes as code 97, but reversed.
Code 144 appears between quotes as code 112, but reversed.
Codes 149-156 appear between quotes as codes 117-124, but reversed.
Codes 158-159 appear between quotes as codes 126-127, but reversed.

Examples:

```
print chr$(130);"this is blinking!"

print chr$(14);"everything is locase"
```

MUSICAL NOTE TABLE

The table below contains the sound register values of six octaves of notes for PAL and NTSC television standards. The sound register values To use the first note in the table (A - sound register value 7) use the 7 as a second number after the SOUND command - SOUND 1,7,30.

NOTE	REGISTER (PAL)	REGISTER (NTSC)	FREQUENCY (HZ)
A	7	7	110.0
#A	64	64	116.6
H	118	118	123.5
C	169	169	130.9
#C	217	217	138.6
D	262	262	146.9
#D	305	305	155.6
E	345	345	164.9
F	383	383	174.7
#F	419	419	185.0
G	453	453	196.0
#G	485	485	207.7
A	516	516	220.0
#A	544	544	233.1
H	571	571	247.0
C	596	597	261.7
#C	620	621	277.2
D	643	643	293.7
#D	664	665	311.2
E	685	685	329.7
F	704	704	349.3
#F	722	722	370.0
G	739	739	392.0
#G	755	755	415.4
A	770	770	440.0
#A	784	784	466.2
H	798	798	493.9
C	810	810	523.3
#C	822	822	554.4
D	834	834	587.4
#D	844	844	622.3
E	854	854	659.3
F	864	864	698.5
#F	873	873	740.0
G	881	881	784.0
#G	889	889	830.7
A	897	897	880.0
#A	904	904	932.4
H	911	911	987.8
C	917	917	1046.6
#C	923	923	1108.8
D	929	929	1174.7
#D	934	934	1244.6
E	939	939	1318.6
F	944	944	1397.0
#F	948	948	1480.0
G	953	953	1568.0
#G	957	957	1661.3
A	960	960	1760.0
#A	964	964	1864.7
H	967	967	1975.6
C	971	971	2093.0
#C	974	974	2217.5
D	976	976	2349.4
#D	979	979	2489.1
E	982	982	2637.1
F	984	984	2793.9
#F	986	986	2960.0
G	988	988	3136.0
#G	990	990	3322.5
A	992	992	3520.0
#A	994	994	3729.3
H	996	996	3951.1

AUTHORS

Janne Peräaho E-mail: amity@surfeu.fi

Pyrytie 11 A 9
90630 Oulu
Finland

Thanks to Mike Dailly for converting the manual to HTML format.

Anders Persson http://listen.to/boray

Thanks to Brian Flood for the Plus/4 and to Janne for the cooperation on this new version of your manual.

ALL DOCUMENTS

This document is part of a document package intended for Plus/4, C16 and C116 users. The documents included are:

- "basic35.pdf", This document - Janne's original manual (more or less).
- "short35.pdf", A basic quick guide.
- "advanced.pdf", The Hardware and Advanced Basic programming.
- "tedmon.pdf", A short description of the built in machine language monitor.

REFERENCES

Commodore 16 Käyttäjän opas, Commodore 16 User Manual, Commodore 64 Käyttäjän opas, Kaikki kuusnelosesta, 3. painos, Commodore Vic-20 Swedish User Manual, Commodore plus/4 and c16 memory map.