# BASIC Keywords

The following is a list of all the BBC BASIC keywords. Those not supported on the Notebook are noted. Differences in operation of some of the commands is also noted. This list is not intended to be a complete reference of the language. There are many good books available on the subject of programming in BASIC and many of these talk specifically about the BBC version of the language. This list may be useful for those who already know how to program or for those inquisitve beginners who would like to experiment.

**ABS**
*var* = ABS (*number*)

Sets *var* equal to the absolute value of *number*. Negative *numbers* are converted to positive. Positive *numbers* are untouched.

**ACS**
*var* = ACS(*number*)

Sets *var* equal to the arc-cosine of the *number*. The result is in radians (which may be converted to degrees using the DEG function).

**ADVAL**

The Notebook has no analogue input port or equivalent. Use of the ADVAL function will result in the error message "Sorry" (error code 255).

**AND**
*var* **=** *number* **AND** *number*

Sets *var* equal to the logical bitwise AND of the two numeric arguments.

**ASC**
*var* = ASC(*string*)

Sets *var* equal to the ASCII value of the first character of the given *string*.

**ASN**
*var* = ASN(*number*)

Sets *var* equal to the Arc Sine of the argument. Result is in radians.

**ATN**
*var* = ATN(*number*)

Sets *var* equal to the Arc Tangent of the given *number*. The result is in radians and may be converted to degrees using the DEG function.

**AUTO**
AUTO *start, step*

Starts generating automatic line numbers at line *start* and goes up by *step*. If *start* and *step* aren't given it starts at 10 and goes up in steps of 10.

**BGET**
*var* = BGET#*number*

Sets *var* equal to the next character from the file that has been opened as *number*. The file can also be "COM:" for the serial port.

**BPUT**
BPUT#*number, value*

Writes the *value* to a file that has been opened as *number*. The file could also be the serial port "COM:" or the parallel port "LPT:".

**CALL**
CALL *address, parameters*

Calls a machine code subroutine. Not for the faint hearted.

**CHAIN**
CHAIN *string*

Loads and then continues on to run the program stored in the file whose name is given in *string*.

**CHR$**
*var* = CHR$(*number*)

Sets a string variable equal to the character whose ASCII code number is *number*.

**CLEAR**
CLEAR

This resets all dynamic variables to the unused condition. The only variables left intact are the static variables A% to Z% and @%.

**CLG**

This statement clears the current graphics window (by default the entire display) to the unlit ('white') state. The graphics cursor is not moved.

**CLOSE**
CLOSE#*number*

Closes the file identified by *number*.

**CLS**

This statement clears the current text window (by default the entire display) to 'space' characters and moves the text cursor to the top left.

**COLOUR**
**COLOR**

The Notebook's (LCD) screen cannot display colours. Use of the COLOUR statement will result in the error message "Sorry" (error code 255). Limited control over text attributes can be obtained with VDU which is explained in the description of VDU.

**COS**
*var* = COS(*number*)

Sets *var* equal to the Cosine of the angle *number* which is specified in radians. The RAD function may be used to convert an angle in degrees to radians.

**COUNT**
*var* = COUNT

Sets *var* equal to the number of characters sent to the display since the last new line.

**DATA**
DATA *constant, constant, constant...*

Used to include constant data within a program which may be used by means of the READ command which will read it into variables.

**DEF**
DEF PROC*name*
DEF FN*name*

Used to begin the definition of a named procedure or function. The following example may give a taste of how this works:

```
10  PROCtest("Hello World")
20  PRINT FN_Average(3,9,14)
30  END
40  :
100 DEF PROCtest(string$)
110 PRINT string$
120 ENDPROC
130 :
200 DEF FNAverage(n1, n2, n3)
210 =(n1+n2+n3)/3
```

**DEG**
*var* = DEG(*number*)

Sets *var* equal to the *number* converted from radians to degrees. In radians a complete circle is equal to 2*PI while in degrees a complete circle is 360 degrees so DEG just divides by 2*PI and multiplies by 360 (which is the same as divide by PI and multiply by 180).

**DELETE**
DELETE *start, finish*

Deletes a range of lines from a program. DELETE 10,100 would remove all lines between 10 and 100 (inclusive). To delete a single line it is easier just to type the line number on its own.

**DIM**
DIM *var, size*

Reserves space for an array of items. For example DIM A(5) would reserve space for 6 items A(0), A(1), A(2)...A(5)

**DIV**
*var* = *number* DIV *number*

Sets *var* equal to the integer result after dividing the first *number* by the second. The remainder is discarded. The function MOD can be used to get the remainder.

**DRAW** *x,y*

Draws a straight line (in 'lit' pixels) between the current position of the graphics cursor and the specified co-ordinates, then moves the graphics cursor to the specified position. The range of co-ordinates corresponding to positions on the screen is 0 to 479 in the x-direction and 0 to 127 in the y-direction. This statement is identical to PLOT 5, x, y.

**EDIT** *number*

A single-line editor is provided, which is entered using the command EDIT *number*. The contents of two or more lines may be concatenated using the syntax EDIT 1,2 but the intermediate line numbers must be edited out, and the original lines deleted, manually. A line may be duplicated by editing only the line number.

**ELSE**
IF *condition* THEN .... ELSE ....

Used to provide an alternative sequence of commands if the *condition* in an IF statement fails.

**END**
END

Marks the point where you would like the program to stop running and return to the BASIC prompt (>). END is not necessary as a program will stop once it has executed the highest line number but END makes things tidy and can be used to END the program early. It also is used to separate the main program from the procedure and function definitions and other subroutines.

**ENDPROC**
ENDPROC

Marks the end of a procedure definition.

**ENVELOPE**

This feature is not supported on the Notebook. Use of the ENVELOPE statement will result in the error message "Sorry" (error code 255).

**EOF**
*var* = EOF#*number*

This function returns TRUE if the file pointer is at the end-of-file identified by *number* and FALSE otherwise. In the special case of the serial port ("COM:") TRUE indicates that there are no input characters waiting while FALSE indicates that one or more characters are waiting at the input.

**EOR**
*var* = *number* EOR *number*

Sets *var* equal to the logical bitwise exclusive OR of the two *numbers*.

**ERL**
*var* = ERL

Sets *var* equal to the number of the last line that caused an error.

**ERR**
*var* = ERR

Sets *var* equal to the number of the last error code. The possible codes are:

| | | |
|---|---|---|
| 1 Out of range | ~~18 Division by zero~~ | ~~33 Can't match FOR~~ |
| 4 Mistake | 19 String too long | ~~34 FOR variable~~ |
| 5 Missing , | 20 Too big | 36 No TO |
| 6 Type mismatch | 21 -ve root | 38 No GOSUB |
| 7 No FN | ~~22 Log range~~ | 39 ON syntax |
| 9 Missing " | ~~23 Accuracy lost~~ | 40 ON range |
| ~~10 Bad DIM~~ | ~~24 Exp range~~ | 41 No such line |
| ~~11 DIM space~~ | 26 No such variable | 42 Out of DATA |
| 12 Not LOCAL | 27 Missing ) | 43 No REPEAT |
| 13 No PROC | ~~28 Bad HEX~~ | 45 Missing # |
| ~~14 Array~~ | 29 No such FN/PROC | |
| 15 Subscript | ~~30 Bad call~~ | |
| 16 Syntax error | ~~31 Arguments~~ | |
| ~~17 Escape~~ | 32 No FOR | |

**ERROR**
ON ERROR GOTO
ON ERROR OFF

Used to trap errors. When an ON ERROR GOTO command is used subsequent errors cause program control to go to the line identified in the GOTO part of the command. ERR and ERL can be inspected to see what caused the error and if it can be corrected.

**EVAL**
*var* = EVAL(BASIC expression in a *string*)

This very powerful command passes the *string* to the BASIC expression handler and then sets *var* equal to the result. A simple 4 line program will turn BASIC into a scientific calculator:

```
10 REPEAT
20  INPUT "Enter command : " e$
30  PRINT EVAL e$
40 UNTIL FALSE
```

When RUN this might give the following:

```
Enter command : SIN(RAD(45))
0.707106781
Enter command : DEG(ATN(SQR(2)))
54.7356103
Enter command : TIME$
Thu.19 Mar 1992,00:27:42
```

**EXP**
*var* = EXP(*number*)

Sets *var* equal to the natural logarithm base (e=2.71828183) raised to the power of *number*. The inverse of this function is provided by LN.

**EXT**
*var* = EXT#*number*

This function returns the size, in bytes, of an opened file. In the special case of the serial and parallel ports ("COM:" and "LPT:") a non-zero returned value indicates that the output port is busy and if written to may result in a "Device fault". A returned value of zero indicates that the output port is ready to receive more characters.

**FALSE**
*var* = FALSE

FALSE is a fixed variable defined as 0. As BASIC uses the value 0 to mean FALSE in conditional tests such as IF and UNTIL you can use FALSE in these situations. For example,

```
REPEAT
PRINT "Hello"
UNTIL FALSE
```

will repeat forever (until Stop is pressed).

**FN**
*var* = FN*name*
DEF FN*name*

Used in both defining and using a named function. See DEF for more details.

**FOR**
FOR *var=start number* TO *finish number* STEP *step value*

Used to start a repetitive loop for a fixed number of iterations. *var* will start at the value *start number* and then, each time a corresponding NEXT instruction is executed *var* will be increased by *step value* (or just 1) until it reaches (or exceeds) *finish number*.

**GCOL**

The Notebook's (LCD) screen cannot display colours. Use of the GCOL statement will result in the error message "Sorry" (error code 255).

**GET**
*var* = GET

Sets *var* equal to the ASCII value of the next key pressed. Waits for a key to be pressed before returning.

**GET$**
*var$* = GET$

Sets the *string* variable *var$* equal equal to the next character key pressed. It waits for a key press before returning.

**GOSUB**
GOSUB *line*

A jump is made to the section of program starting at *line*. When the next RETURN command is executed control returns to the statement after the GOSUB command.

**GOTO**
GOTO *line*

Control is transferred to the *line* identified in the GOTO command. To make programs easy to read the use of excessive GOTO commands should be avoided. It is far better to change the flow of a program using the FOR...NEXT, REPEAT...UNTIL, DEF PROC and GOSUB structures.

**HIMEM**
HIMEM = *number*
*var* = HIMEM

Can be used either to set a new high address for the top of BASIC's program memory or to find out what it is currently set to. It is unwise to change this unless you are sure you know what you are doing as you may crash the machine leading to the need to completely reset it and lose all your documents. It is OK to reduce HIMEM but do not increase it above its initial value.

**IF**
IF *condition* THEN

Used to conditionally execute statements. The condition is tested and if it results in a TRUE (-1) value the statements after THEN are executed.

**INKEY**
*var* = INKEY(*time*)

Sets *var* equal to the ASCII value of the next key pressed. Unlike GET it only waits for the length of time given by *time* in centiseconds. If no key is pressed it returns -1. Use of INKEY with a negative argument to test the state of each key independently is not supported.

**INKEY$**
*var$* = INKEY$(*time*)

Waits for *time* 1/100ths of a second for a key to be pressed and returns the character in *var$*. If no key is pushed in time it sets *var$* to the null (empty) string.

**INPUT**
INPUT "*prompt text*", *var*

Stops and displays the *prompt text* and then sets *var* equal to the users response. *var* can also be a string variable to allow the user to enter text as well as numbers.

**INPUT LINE**
INPUT LINE *var$*

Allows the user to type a string of text including commas, quotes and leading spaces and assigns this to *var$*.

**INPUT#**
INPUT#*number*, *var*

Inputs variable *var* from the file identified by *number*.

**INSTR**
*var* = INSTR(*string*, *string to find*, *number*)

The first named *string* is searched to see if it contains the *string to find* and if so *var* is set to the position in the string where it occurs. The search can be started part way into the string by giving a *number*.

**INT**
*var* = INT(*number*)

Converts a real *number* to a lower integer.

**LEFT$**
*var$* = LEFT$(*string$*, *number*)

Takes the *number* of leftmost characters from *string$* and assigns them to *var$*.

**LEN**
*var* = LEN(*string$*)

Sets *var* equal to the number of characters in the given *string*.

**LET**
LET *var* = *value*

LET assigns a *value* to a variable (either number or string). In fact LET is optional and need not be used. So

```
LET X = X + 3
```

and

```
X = X + 3
```

are exactly equivalent.

**LINE**
INPUT LINE

See INPUT LINE.

**LIST**
LIST *number*, *number*

Lists a program. If a single *number* is given then only that line is shown. If both *numbers* are given then all lines between the two are shown. While a program is listed you can press Stop to pause the listing. Press it again to stop.

**LISTO**
LISTO *number*

LISTO is used before the LIST command to control how the subsequent listing is formatted. The *number* affects how the listing is formatted. Valid *numbers* are 0 to 7. LISTO 7 gives the easiest to follow listing.

**LN**
*var* = LN(*number*)

Sets *var* equal to the natural logarithm of the *number*. Natural logs are to the base e (=2.71828183). The inverse of LN is EXP.

**LOAD**
LOAD *prog_name*

The *prog_name* (in quotes or contained in string variable) is the name of a document that contains a program to load.

**LOCAL**
LOCAL *var*

Specifies a variable that is only local within a procedure or function declaration. The value is lost (undefined) outside of the proc/fn.

**LOG**
*var* = LOG(*number*)

Sets *var* equal to the base 10 logarithm of *number*. There is no inverse function of LOG as such because the equivalent is to use 10^*number* (ie 10 raised to the power of a *number*).

**LOMEM**
LOMEM = *var*
*var* = LOMEM

May be used to read and set the point in memory where dynamic data structures will be placed. It would be unwise to change this unless you are absolutely certain that you know what you are doing.

**MID$**
*var$* = MID$(*string$*, *start*, *length*)

Sets *var$* equal to a string of characters taken from *string$* starting at position *start* for *length* characters.

**MOD**
*var* = *number* MOD *number*

Divides the first *number* by the second and sets *var* equal to the remainder. See also DIV which sets *var* equal to the integer result of dividing one number by another.

**MODE**

This feature is not supported on the Notebook. Use of the MODE statement will result in the error message "Sorry" (error code 255).

**MOVE**
MOVE x,y

Moves the graphics cursor to the specified co-ordinates, but does not affect what is displayed. The range of co-ordinates corresponding to positions on the screen is 0 (left) to 479 (right) in the x-direction and 0 (bottom) to 127 (top) in the y-direction. This statement is identical to PLOT 4, x, y.

**NEW**
NEW

Clears the current program from memory. If you use this accidentally you can immediately use OLD to recover it but OLD will no longer function once you start to enter new program lines or set new variables.

**NEXT**
NEXT *var*

Used to mark the end of a FOR loop and cause a jump back to the statement after FOR until the loop variable has reached its upper limit. The *var* name can be used to make sure a jump is made back to the correct FOR command.

**NOT**
*var* = NOT *number*

Sets *var* equal to the bit by bit binary inversion of *number.*

**OLD**
OLD

Used to recover a program immediately after the accidental use of NEW.

**ON**
ON *var* GOTO *line, line...*
ON *var* GOSUB *line, line...*
ON *var* PROC*one*, PROC*two*, ...

Can be used to goto or gosub a number of different lines depending on the value of *var.* Can also be used to call different named procedures dependent on the value of *var.*

**OPENIN**
*var* = OPENIN(*string*)

The document/file whose name is given by *string* is opened for input (reading) and a file number is returned in *var.* This may be used in the various file reading commands such as BGET# and INPUT#. If the filename given is "COM:" then input will be read from the serial port.

**OPENOUT**
*var* = OPENOUT(*string*)

The document/file whose name is given by *string* is opened for output (writing) and a file number is returned in *var.* This may be used in the various file writing commands such as BPUT#. If the filename given is "COM:" then output will go to the serial port. If the name is "LPT:" then the output will be to the parallel printer port.

**OPENUP**
*var* = OPENUP(*string*)

Has the combined effect of OPENIN and OPENOUT. The device or file is opened for reading and writing. Can be used with files and the serial port "COM:"

**OR**
*var* = *number* OR *number*

Sets *var* equal to the result of the logical bitwise OR of the two *numbers.*

**OSCLI**
OSCLI(*string*)

The *string* is passed to the operating system to be executed. This can be one of the star commands such as *CAT. That is, OSCLI("CAT"). Notice that * is not included.

**PAGE**
PAGE = *var*
*var* = PAGE

PAGE can be used to read or set the starting address of the current program area. It would be extremely unwise to change this value unless you are totally certain you know what you are doing.

**PI**
*var* = PI

Sets *var* equal to the value of π 3.14159265, the ratio of a circle's circumference to its diameter.

**PLOT**
PLOT *n*,x,y

A multi-purpose plotting statement, whose effect is controlled by the number *n*. *x* must be in the range 0 to 479 and *y* 0 to 127. In the following "relative" means that (x, y) are added onto the current graphics cursor position to determine the destination. When "absolute" co-ordinates are used they are always specified with relation to the origin of the graphics screen at (0, 0).

| *n* | **Action** |
|---|---|
| 0 | Moves the graphics cursor relative to the last point. |
| 1 | Draws a line, in 'black', relative to the last point. |
| 2 | Draws a line, in 'inverse', relative to the last point. |
| 3 | Draws a line, in 'white', relative to the last point. |
| 4 | Moves the graphics cursor to the absolute position x,y. |
| 5 | Draws a line, in 'black', to the absolute position x,y. |
| 6 | Draws a line, in 'inverse', to the absolute position x,y |
| 7 | Draws a line, in 'white', to the absolute position x,y. |
| 8-15 | As 0-7, but plots the last point on the line twice (i.e. in the 'inverting' modes omits the last point). |
| 16-31 | As 0-15, but draws the line dotted. |
| 32-63 | As 0-31, but plots the first point on the line twice (i.e. in the 'inverting' modes omits the first point). |
| 64-71 | As 0-7, but plots a single point at x,y. |
| 72-79 | Draws a horizontal line left and right from the point x,y until the first 'lit' pixel is encountered, or the edge of the window. This can be used to fill shapes. |
| 80-87 | Plots and fills a triangle defined by the two previously visited points and the point x,y. |
| 88-95 | Draws a horizontal line to the right of the point x,y until the first 'unlit' pixel is encountered, or the edge of the window. This can be used to "undraw" things. |
| 96-103 | Plots and fills a rectangle whose opposite corners are defined by the last visited point and the point x,y. |

**POINT**
*var* = POINT (x,y)

This function sets *var* to the state of the pixel at the specified location, as 0 (unlit) or 1 (lit). If the specified point is outside the graphics window (taking into account the position of the graphics origin) the value -1 is returned. *x* is in the range 0..479 and *y* is in the range 0..127

**POS**
*var* = POS

This function sets *var* equal to the horizontal position (column) of the text cursor with respect to the left-hand edge of the current text window, in the range 0 to 79.

**PRINT**
PRINT *var*

Prints the contents of a variable or variables and fixed text on the screen. The line of items following PRINT is passed to BASIC's expression evaluator before output is produced. A tilde character "~" can be used before numeric items that should print in hexadecimal. Commas in the print list cause output to start at the next tab stop. Semicolons mean the items follow on immediately adjacent. A single apostrophe forces printing to start on a new line.

Print format can be controlled by setting the variable @% before printing but there are too many options to describe here. The functions TAB(x,y) and SPC(*number*) can be used in a print statement to either position the cursor at location (x,y) or at a fixed *number* of spaces from the current position.

**PROC**
PROC*name*

Used to invoke a named procedure that is defined using DEF PROC.

**PTR**
PTR#*number* = *var*
*var* = PTR#*number*

Allows the random access pointer of file *number* to be read or set.

**PUT**
PUT *port*, *var*

Outputs a value to an I/O *port* address. It would be very unwise indeed to experiment with this command as you will almost certainly crash the Notebook necessitating a reset which will lose all stored data.

**RAD**
*var* = RAD(*number*)

Sets *var* equal to the value of *number* converted from degrees to radians.

**READ**
READ *var*

Reads data from a DATA statement and assigns it to *var*.

**REM**
REM *comment*

Allows comments to be added to programs. Anything following REM is considered a remark and will be ignored. It is good practice to include comments in programs so that at a later date you will understand what a particularly complicated line really does! It can also be used to temporarily disable a command while testing a program

**RENUMBER**
RENUMBER *start*, *step*

Renumbers a program starting at line 10 and going up in steps of 10. If *start* and *step* are given then numbering will begin at *start* and go up in units of *step*.

**REPEAT**
REPEAT

Used to begin a loop that ends with the command UNTIL.

**REPORT**
REPORT

This prints the error message associated with the last error that occurred and is usually used in an ON ERROR GOTO trap to print an error message when your program determines that the error which has occurred is not one that it can cope with.

**RESTORE**
RESTORE *line*

When using READ to read data from DATA statements, RESTORE can be used to set the reading pointer back to a specified line so that data can be re-read.

**RETURN**
RETURN

Used at the end of a section of program that has been jumped to by the GOSUB command. Control returns to the statement after GOSUB.

**RIGHT$**
*var$* = RIGHT$(*string$*, *number*)

Sets *var$* equal to the string of characters taken from the rightmost end of *string$* and of length *number*.

**RND**
*var* = RND(*number*)

Sets *var* equal to a random number between 1 and *number*. If *number* is not given then the *var* is between 1 an &FFFFFFFF. If the given *number* is negative then the random number generator is set to a value based on that number and that *number* is returned in *var*.

**RUN**
RUN

Starts running the program currently held in memory.

**SAVE**
SAVE *prog_name*

The current program in memory is saved to a document called *prog_name* (this may be either a name in quotes or a *string* variable

that contains a name to use). It is very important to SAVE a program you are working on before switching away from BASIC (or typing *QUIT) as the program is not automatically saved.

**SGN**
*var* = SGN (*number*)

Sets *var* equal to -1 if the *number* is negative, +1 if it is positive or 0 if *number* is zero.

**SIN**
*var* = SIN(*number*)

Sets *var* equal to the Sine of the given angle which must be specified in radians. The RAD function can be used to change an angle in degrees into radians before using the SIN function.

**SOUND**
SOUND *channel*, *volume*, *pitch*, *duration*

SOUND will makes a sound on the Notebook's speaker. The Notebook has two sound channels so *channel* must be either 1 or 2. The volume is not variable so the *volume* parameter is ignored. The *pitch* specifies the note to be played. A value of 100 is middle C. Although each step of the pitch parameter should change the pitch by a quarter semitone, the Notebook can only play notes in steps of a semitone so there is no point using values that are not a multiple of four. A *pitch* value of 0 will switch a sound channel off.

The *duration* is given in twentieths of a second in the range 0 to 254 The value -1 or 255 causes an indefinite sound, which can be stopped only by issuing another SOUND statement or by pressing the Stop key. So, for example:

```
SOUND 1, 0, 136, 20
```

Will play the A above middle C on channel 1 for 1 second (20/20ths).

To play a two note chord, use a SOUND command with a duration of -1 to start the first note on channel 1 and then use a second SOUND command to play the other note "on top" on channel 2. Sounds can be made to stop playing by using another SOUND command with a duration of 0.

**SPC**
PRINT SPC(*number*)
INPUT SPC(*number*)

When used in either a PRINT or INPUT command it prints *number* spaces before any following text.

**SQR**
*var* = SQR(*number*)

Sets *var* equal to the square root of *number*.

**STEP**
FOR *var*=*start* TO *finish* STEP *step*

Allows a FOR..NEXT variable to be increased (or decreased) in steps other than one.

**STOP**
STOP

Just like END it stops a program running but prints a message to say where the program stopped. Liberal use of STOP commands can help when developing a program to trace the flow of execution.

**STR$**
*var$* = STR$(*number*)

Sets a *string* to be equal to a *number* in the same format that it would be printed in. If a tilde is included between the $ and open parenthesis the *number* will be converted to hexadecimal.

**STRING$**
*var$* = STRING$(*number*, *string*)

Sets *var$* equal to *number* repetitions of *string*.

**TAB**
PRINT TAB(x,y)
INPUT TAB(x,y)

Used to arrange for the printed output of a PRINT or INPUT command to appear on the screen at location (x,y) with respect to the current text window.

**TAN**
*var* **= TAN(***number***)**

Sets *var* equal to the Tangent of the angle *number*. This must be specified in radians. To use degrees use the RAD function to convert the value from degrees to radians before using TAN.

**THEN**
IF *condition* THEN

Introduces the statements in an IF command that should be executed if the condition is met. The use of THEN is optional but makes programs easier to read.

**TIME**
TIME = *var*
*var* = TIME

A variable that can be set and read to measure elapsed time. It increases once every 1/100th of a second. Typically this is used to measure a fixed amount of elapsed time. For example:

```
TIME = 0
REPEAT
UNTIL TIME > 1000
```

would pause for approximately 10 seconds.

**TIME$**
*var$* = TIME$
TIME$ = *var$*

Sets *var$* equal to a *string* which contains the current date and time in a fixed format. MID$ can be used to pick selected fields from this. It is also possible to set the date and time stored in the Notebook by setting the TIME$ variable. The format of the returned string is "*Day.dd Mon yyyy,hh:mm:ss*" where *Day* is the day of week, *dd* the day of month, *Mon* the month, *yyyy* the year, *hh* the hour (00-23), *mm* the minute and *ss* the second. The time, date or both time and date may be set by including the appropriate fields. When setting the clock the day of week is ignored and may be omitted.

**TO**
FOR *var* = *start* TO *finish*

Used in a FOR statement to divide the starting value from the end value of the loop variable.

**TRACE**
TRACE ON
TRACE OFF
TRACE *number*

The command TRACE ON will cause BASIC to print the number of each line it executes in square brackets to allow the flow of execution to be followed. Tracing can be turned off using the command TRACE OFF. If the command TRACE *number* is used then only line numbers below *number* will be printed. By placing all subroutines at high numbered lines and the main program in the low numbered lines you can arrange to only show tracing of the main part of the program.

**TRUE**
*var* = TRUE

Sets *var* to be equal to the value -1 that BASIC understands as TRUE in IF and UNTIL expressions

**UNTIL**
UNTIL *condition*

Ends a loop started by the REPEAT command. As long as the condition is not met a jump will be made back to the statement after REPEAT.

**USR**
*var* = USR(*number*)

Calls a machine code routine at address *number* and returns the value of the HL and HL' registers to the named variable. Not a command for the uninitiated in the black art of machine code programming.

**VAL**
*var* = VAL(*string*)

Converts as much of a string that can be interpreted as a number into a numeric value and assigns it to *var*.

**VDU**
VDU *number, number, ....*

Passes the elements of the list to the VDU emulator (see full description below). Items terminated by a semicolon are sent as 16-bit values, LSB first.

All console output is passed to a software emulator of the BBC Micro's VDU drivers. VDU codes perform a function similar to those of the BBC Micro, consistent with the hardware and Operating System differences:

| | |
|---|---|
| VDU 0 | Ignored |
| VDU 1,n | The following byte is sent to the printer, if enabled (with VDU 2). If the printer is not enabled, the byte is ditched. Any 8-bit value (0- 255) can be sent. This works even when the VDU is disabled with VDU 21. |
| VDU 2 | Enables the printer. Subsequent characters are sent both to the screen and to the printer. The only control characters sent to the printer are BEL (7), BS (8), HT (9), LF (10), VT (11), FF (12) and CR (13). Bytes which are parameters for VDU commands are not sent to the printer, e.g. VDU 27,13 does not send a carriage return to the printer. |
| VDU 3 | Disables the printer. Cancels the effect of VDU 2. |
| VDU 4 | Causes the text cursor to be displayed. |
| VDU 5 | Causes the text cursor to be hidden. |
| VDU 6 | Enables the screen display. Cancels the effect of VDU 21. |
| VDU 7 | Causes a "beep". |
| VDU 8 | Moves the text cursor left one character. If it was at the left edge of the window, it is wrapped to the end of the previous row (right-hand edge of window). If it was also on the top row of the text window, it is moved to the bottom row. |
| VDU 9 | Moves the text cursor right one character. If it was at the right hand edge of the window, it is wrapped to the beginning of the next row (left-hand edge of window). If it was also on the bottom row of the text window, it is moved to the top row. |
| VDU 10 | Moves the text cursor down one row. If it was on the bottom row of the text window, the window scrolls up. |
| VDU 11 | Moves the text cursor up one row. If it was on the top row of the text window, it is moved to the bottom row. |
| VDU 12 | This is identical to CLS in BASIC. It clears the text window to space characters and moves the text cursor to the 'home' position (top-left corner of the text window). |
| VDU 13 | Moves the text cursor to the left-hand edge of the window, but does not move it vertically. |
| VDU 14 | Enables inverse text. |
| VDU 15 | Disables inverse text. Cancels the effect of VDU 14. |
| VDU 16 | This is identical to CLG in BASIC. It clears the graphics window to unlit ('white') pixels. The graphics cursor is not moved. |
| VDU 17 | Enables bold text. |
| VDU 18 | Disables bold text. Cancels the effect of VDU 17. |
| VDU 19 | Enables underlined text. |
| VDU 20 | Disables underlined text. Cancels the effect of VDU 19. |
| VDU 21 | Disables VDU output. All subsequent VDU commands except 1 to 6 are ignored. If the printer is enabled, VDUs 7,8,9,10,11,12 and 13 will still be sent to the printer. |
| VDU 22 | Ignored |
| VDU 23 | Ignored |

VDU 24,leftx;bottomy;rightx;topy;

Sets the graphics window. Horizontal (x) co-ordinates are in the range 0 (left) to 479 and vertical (y) co-ordinates in the range 0 (bottom) to 127.

VDU 25, n, x; y;

This is identical to PLOT n, x, y in BASIC. See PLOT for more details.

| | |
|---|---|
| VDU 26 | Resets the text and graphics windows to their default positions (filling the whole screen), homes the text cursor, moves the graphics cursor to 0,0 and resets the graphics origin to 0,0. |
| VDU 27,n | Sends the next byte to the screen without interpreting it as a control code. Allows graphics characters corresponding to VDU 0-31 and VDU 127 to be displayed. Acts in a similar way to VDU 1 for the printer. |

VDU 28, leftx, bottomy, rightx, topy

Sets the text window. Horizontal (x) co-ordinates are in the range 0 (left) to 79 and vertical (y) co-ordinates in the range 0 (top) to 7.

| | |
|---|---|
| VDU 29, x; y; | Moves the graphics origin to the specified co-ordinates. Subsequent graphics co-ordinates are with respect to this position. |
| VDU 30 | Homes the text cursor, to the top-left corner of the text window. |
| VDU 31, x, y | Identical to PRINT TAB(x, y); in BASIC. Positions the text cursor according to the next two bytes. The co-ordinates are with respect to the edges of the current text window. |
| VDU 127 | Backspaces the cursor by one position and deletes the character there. |

**VPOS**
*var* = VPOS

Sets *var* equal to the vertical position (row) of the text cursor with respect to the top of the current text window, in the range 0 to 7.

**WIDTH**
WIDTH *number*

Sets the width of print zones. A value of zero will stop it taking any action.

## Operating System Commands

The following Operating System commands are implemented. They may be accessed directly (e.g. *BYE) or via the OSCLI statement (OSCLI "BYE").

***BYE**
***QUIT**

Exits from BASIC and returns control to the Operating System.

***CAT**
***DIR**

Lists a catalogue of all the stored files.

***DELETE** *filename*
***ERASE** *filename*

Deletes the specified file.

***ESC (ON|OFF)**

Enables or disables the abort action of the [Stop] key which is known in BASIC as the ESCape key; after *ESC OFF the [Stop] key simply returns the ASCII code ESC (27). *ESC ON, or *ESC, restores the normal action of the [Stop] key.

***EXEC** *filename*

Accepts console input from the specified file instead of from the keyboard (note that GET and INKEY always read from the keyboard).

***KEY** *n* **(***string***)**

Redefines a key to return the specified string. The key number n is from 0 to 127, where 41 to 66 correspond to [Symbol][^] to [Symbol][Z] respectively. The string may contain the "escape" symbol | in order to insert non-printing characters. For example, |M indicates [↵], |? indicates DEL, || indicates the character | itself and |! causes bit 7 of the following character to be set. If the string is enclosed in quotes (which is optional) |" allows the character " to be included.

***LOAD** *filename aaaa*

Loads the specified file into memory at hexadecimal address aaaa. The load address must be specified, and point to a valid memory location.

***PRINTER** *n*

Selects the printer as parallel (*n*=0) or serial (*n*=1).

***RENAME** *oldfile newfile*

Renames the file *oldfile* as *newfile*.

***SAVE** *filename aaaa bbbb*
***SAVE** *filename aaaa* **+***llll*

Saves a specified range of memory to a file. The address range is specified either as start address *aaaa* and end address +1 *bbbb* or as start address *aaaa* and length *llll*.

***SPOOL (***filename***)**

Copy all subsequent console output to the specified file. If the *filename* is omitted, any current spool file is closed and spooling is terminated.

***|** *comment*

This is a comment line. Anything following the | is ignored.

**Note**: To type the | symbol hold down [^] and press [\]

**BASIC and the Printer**

To list a program on the printer type:

```
VDU 2
LIST
```

When listing is finished type VDU 3 to cancel printer output.

To have the output of a PRINT command in a program only output on the printer use something like the following:

```
10 VDU 2:VDU 21
20 PRINT "This only appears on the printer"
30 VDU 3:VDU 6
```

## Operating System error messages

These (trappable) errors are related to operating system functions:

| | |
|---|---|
| Access denied (189): | An inappropriate operation was attempted on a device (e.g. reading from the parallel port). |
| Bad command (254): | A star command was invalid or incorrectly formed. |
| Bad key (251): | An attempt to define a function key string failed. |
| Bad string (253): | A string was too long, or had unmatched quotes. |
| Channel (222): | The channel number passed to a filing function was invalid |
| Close error (200): | An error occurred when trying to close a file. |
| Device fault (202): | A time-out occurred when reading or writing a device. |

| | |
|---|---|
| File creation error (190): | An OPENOUT, SAVE, *SAVE or *SPOOL failed, because the specified file could not be created (e.g. too many files). |
| File exists (196): | A *RENAME command specified the new name as the name of an existing file. |
| File not found (214): | A LOAD, *LOAD, *EXEC, *DELETE or *RENAME failed, because the specified file did not exist. |
| File write error (198): | An error occurred when writing a file with SAVE, *SAVE, PRINT#, BPUT# or *SPOOL, e.g. because of insufficient memory. |
| Too many open files (192): | An attempt was made to exceed the maximum number of open files (7). |