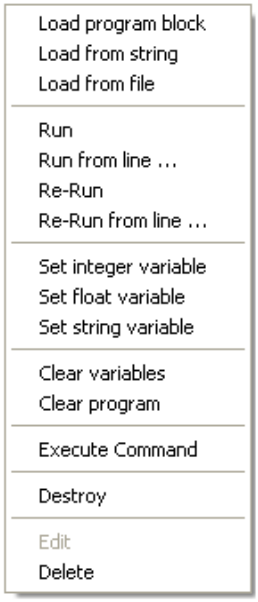
BASIC OBJECT

ACTIONS



Load

program block: load the internal basic program
from string: load a source from a string
from file: load from file selector, program must have .BAS extension.

Run

All: From start
from line: Set the staring line number

Re-Run

All: From start
from line: Set the staring line number

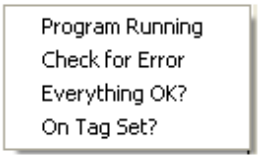
Set Value

Integer: Set the integer value for an internal variable, remember those variables are case sensitive and may have max 20 chars length.
Float: Set a float value for a variable.
String: Set string content for a string variable.

Clear

Variables
Program: Clear variable and program

CONDITIONS



Program Running

An Always & notable condition, return true when block of program is running.

Check for Error

An Error occurred during last running or parse state.

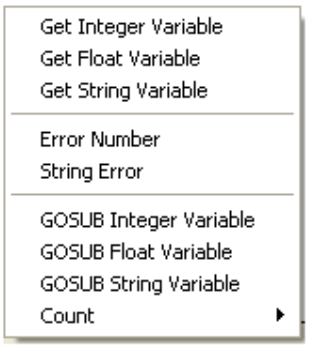
Everything Ok

All Ok during last running state.

On Tag Set

If in block of program a MMFTAG is issued and the name correspond (no case sensitive) this condition will be executed.

EXPRESSIONS



Get

Integer: Get the integer value for an internal variable.
Float: Get a float value for a variable.
String: Get string content for a string variable.

Error

Number: number of last error.
String: text of last error occured.

Gosub

Execute the code from line X and return value variable in RETMMF command.

Integer: Get the integer value for an internal variable.
Float: Get a float value for a variable.
String: Get string content for a string variable.

COMMANDS

LIST line(s)

List the specified program lines. For example,

LIST 10, 100-200

lists line 10, and lines 100 through 200, inclusive.

RUN [line]

Begin execution of the program at the first line, or at the specified line. All variables are cleared.

RUN file[,line]

Load and run a program. For example,

RUN "FOO ", 30

loads a program from the file FOO.BAS and begins execution at line 30.

RERUN [line]

Begin execution of the program at the first line, or at the specified line.

NEW

Erase the program in memory.

CLEAR

Clean all variables.

LOAD file

Load a program into memory. The program previously in memory is erased. The file name should be in quotes; a .BAS extension is automatically added. Files contain ASCII listings of the programs. All lines in the file must begin with a line number, but line numbers do not need to be in increasing order.

MERGE file

Load a program into memory. The previous program remains in memory; if a line exists in both programs, the newly loaded line is kept.

SAVE file

Save the program in memory to a file.

DEL line(s)

Delete the specified program lines. Line numbers may be separated by commas and dashes as in LIST. If used inside a program, DEL will terminate execution only if it deletes the line on which it appears.

RENUM [start[,inc]]

Renumber program lines. By default, the new sequence is 10,20,30,... The first argument is a new initial line number; the second argument is the increment between line numbers.

STATEMENTS

REM comment

A remark; ignored. Comments may contain any characters except that REM can not be immediately followed by an alphanumeric character.

[**LET**] var = expr

Assign a value to a variable. Variable names contain up to 20 significant characters, consisting of upper- and lower-case letters, digits, underscores, and dollar signs. Variable names are case-sensitive. Variables hold real numbers normally, or strings of up to 255 characters if their names end with $.

Examples:

```
LET X=20
X$="FOO"
X$=X$+"BAR"
```

DIM var(dimensions), ...

Allocate memory for arrays. Arrays may have up to 4 dimensions, ranging from 1 to the value specified in the DIM statement. The same name must not be used for both a simple variable and an array.

If an array is used before it is dimensioned, each dimension is set to 10.

Example:

```
INPUT "How many elements? "; x
DIM array(x,1)
FOR i=1 TO x : INPUT array(i,0), array(i,1) : NEXT
```

PRINT items

Print the items on the screen. Items may be either numeric or string expressions, and may be separated by commas, semicolons, or nothing.

Print will be present in a message box, the title and icon of this message box can be change using the following commands:

MSGTIT [prompt] vars

MSGICON vars o value

MB_OK	0
MB_OKCANCEL	1
MB_ABORTRETRYIGNORE	2
MB_YESNOCANCEL	3
MB_YESNO	4
MB_RETRYCANCEL	5
MB_ICONHAND	16
MB_ICONQUESTION	32
MB_ICONEXCLAMATION	48
MB_ICONASTERISK	64
MB_DEFBUTTON1	0
MB_DEFBUTTON2	256
MB_DEFBUTTON3	512
MB_DEFBUTTON4	768

Numbers are normally terminated by spaces. To avoid this space, convert the number to a string with STR$.

The line is terminated by a CR/LF, unless the item list ends with a comma or semicolon.

The word PRINT may be abbreviated as a question mark.

Examples:

```
PRINT "1+2=", 1+2
PRINT X$ "=" Z$;
? x; y+z
```

INPUT [prompt;] vars

If a prompt string is given, it is printed. Otherwise, a question mark is printed. The computer then waits for values for each variable to be entered. If several variables are listed, their names must be separated by commas.

If the variables are numeric, their values may be entered on separate lines, or combined with commas. Any numeric expression is a valid response.

If the variables are strings, each string is typed on a separate line. The characters typed are copied verbatim into the string.

String and numeric variables may be not mixed in a single INPUT statement.

Examples:

```
INPUT X$
INPUT "Type 3 numbers: "; X, Y, Z
```

GOTO line

Begin executing statements at the specified line. The line number may be any numeric expression.

The word GO TO may be used instead of GOTO if preferable.

IF condition THEN line/statements ELSE line/statements

If the condition is true (i.e., the numeric expression has a non-zero value), the statements following the word THEN are executed. Otherwise, the statements following ELSE are executed. If there is no ELSE clause, execution proceeds to the next line in the program.

A line number may be used after either THEN or ELSE, for an implied GOTO statement.

END

Terminate the program. An END statement is not required.

STOP

Terminate the program with an identifying "Break" message.

FOR var = first **TO** last [**STEP** inc]
{statements}
NEXT [var]

Execute {statements} repeatedly while the variable counts from "first" to "last," incrementing by 1, or by the STEP value if given. If the STEP value is negative, the variable counts downward.

If "first" is greater than "last" (or less than if STEP is negative), execution proceeds directly to the NEXT statement, without executing the body of the loop at all.

The variable name is optional on the NEXT statement.

WHILE [condition]
{statements}
WEND [condition]

Execute {statements} repeatedly until the WHILE condition (if given) becomes false, or until the WEND condition becomes true. This structure can emulate Pascal's WHILE-DO and REPEAT-UNTIL, or even both at once. If no conditions are given, the loop will never terminate unless the Evil GOTO is used.

GOSUB line
RETURN

Execute the statements beginning at the specified line, then when RETURN is reached, return to the statement following the GOSUB.

READ vars
DATA values
RESTORE line

Read numeric or string values from the DATA statements. Reading begins at the first DATA statement in the program and proceeds to the last. Reading past the end the last DATA statement generates an error.

The DATA values must be either numeric or string expressions, according to the type of variable being read. Reading the wrong kind of expression produces a Syntax Error.

The RESTORE statement causes the next READ to re-use the first DATA statement in the program, or the first DATA statement on or after a particular line.

ON expr **GOTO** line, line, ...
ON expr **GOSUB** line, line, ...

If the expression's value, rounded to an integer, is N, go to the Nth line number in the list. If N is less than one or is too large, execution continues at the next statement after the ON-GOTO or ON-GOSUB.

MMFTAG(vars)
Generate an event call like vars (string) in MMF event list (max 31 per run)

Example:

```
10 ….
20 MMFTAG(“COUNTER”)
30 …
```

IN MMF

Event List

ONTAG("COUNTER") -> Actions

RETMMF vars

On Gosub from expression you can get a return for MMF, int, float or string, this is as required.

Example:

```
10 ….
20 ….: A$="hello"
30 RETMMF A$
```

matching MMF should be

```
ONGOSUB$(10)
```

NUMERIC EXPRESSIONS

x AND y

Logical AND of two integers.

x OR y

Logical OR of two integers.

x XOR y

Logical XOR of two integers.

NOT x

Logical complement of an integer.

x+y, x-y, x*y, x/y, x^y, -x

Typical floating-point arithmetic operations.

x=y, x<y, x>y, x<=y, x>=y, x<>y

Comparisons; result is 1 if true, 0 if false.

x MOD y

Modulo of two integers.

SQR x

Square of X. Note that parentheses are not required if a function's argument is a single entity; for example, SQR SIN X needs no parentheses, but SQR(1+X) does.

SQRT x

Square root of X.

SIN x, COS x, TAN x, ARCTAN x

Typical trig functions, in radians.

LOG x, EXP x

Natural logarithm and e the power X.

ABS x

Absolute value of X.

SGN x

Sign of X: 1 if X is positive, 0 if zero, -1 if negative.

INT x

Convert double to Integer value, rounding all decimal.

CEILING x

Returns the next highest integer that is greater than or equal to the specified numeric expression.

FLOOR x

Returns the nearest integer that is less than or equal to the specified numeric expression.

VAL x$

Value of the expression contained in the string X$. For example, VAL "1+2" yields 3. X$ may be a single string literal, variable, or function, or a string expression in parentheses.

OCCUR x$, y$

Returns the number of times a character expression occurs within another character expression.

ATC x$, y$ [,nOccurence]

Returns the beginning numeric position of the first occurrence of a character expression within another character expression, without regard for the case of these two expressions.

CRLF$

Returns a string with only CR/LF character as expression.

ASC x$

ASCII code of the first character in X$, or 0 if X$ is null.

LEN x$

Number of characters in X$.

Precedence: Parentheses
Functions (incl. NOT and unary minus)
^
*, /, MOD
+, -
=, <, >, <=, >=, <>
AND
OR, XOR

LEFT$ x$

Returns a specified number of characters from a character expression, starting with the leftmost character.

RIGHT$ x$

Returns the specified number of rightmost characters from a character string.

LTRIM$ x$

Returns the specified character expression with leading blanks removed.

RTRIM$ x$

Returns a character string that results from removing the trailing blanks from a character expression.

LOWER$ x$

Returns a specified character expression in lowercase letters.

UPPER$ x$

Returns the specified character expression in uppercase.

PROPER$ x$

Returns from a character expression a string capitalized as appropriate for proper names.

STRING EXPRESSIONS

"string" or 'string'

String literal. Single quotes are converted to double quotes internally.

x$+y$

Concatenation. Result must be 1024 characters or less.

x$=y$, x$<y$, x$>y$, x$<=y$, x$>=y$, x$<>y$

String comparisons; result is 1 if true, 0 if false.

STR$(x)

The number X expressed as a string of digits. No leading or trailing spaces are included; scientific notation is used if the absolute values is greater than 1E12 or less than 1E-2.

CHR$(x)

The character whose ASCII code is X.

MID$(x$, y)
MID$(x$, y, z)

(Parentheses required.) The substring consisting of the first Z characters starting at position Y of string X$. Position 1 is the first character of the string. If Z is omitted, 1024 is used, i.e., the entire right part of the string.

CONVENTIONS

Multiple statements may be written on a line, separated by colons:

```
10 INPUT X : PRINT X : STOP
```

There is actually no difference between commands and statements; both can be used in or out of programs at will. Certain commands, such as NEW, will, of course, halt program execution.

Line numbers may be any integer from 1 to 10000.

Keywords must be written in all upper- or all lower-case; they are always converted to upper-case internally. Spaces are ignored in the input except between quotes. Square brackets are converted to parentheses. Missing closing quotes at the end of the line are added, as in the command:

```
SAVE "PROGRAM
```