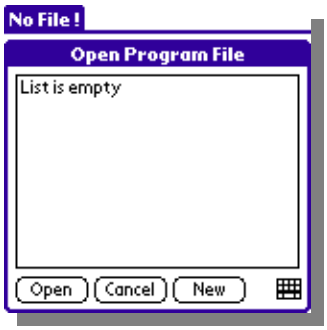
Table of Contents

FastBasic Quick Start

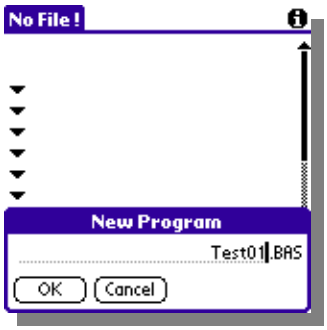
When you start FastBasic the first time, you have a screen similar to the one you can see here:



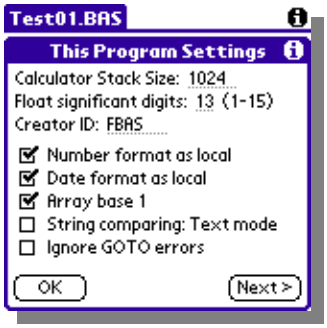
We will create a new program that will show all available characters:

```
FOR n = 0 TO 255
PRINT CHR$(n)
NEXT n
```

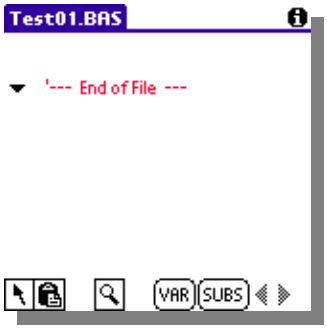
To create a new program, tap the button ***New***.



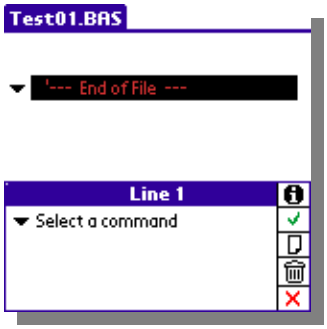
This will open a new window asking for the name of the new program.
Enter the name you prefer, for instance ***Test01***.
Don't include the extension ***.BAS*** because it is automatically added.
Tap the button ***OK***.



Now, it appears another window with the settings for the program.
Tap the button ***OK*** to set the default options.
You can tap the ***"i"*** button to show the related help page.

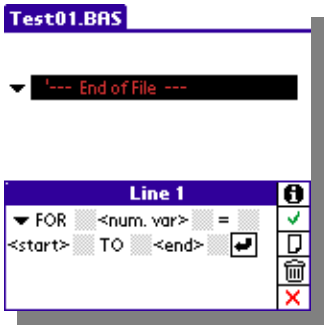


Ok, we have now an empty program:
The only line we can see simply says "End of File".
Tap this line to add the first line of our first program.

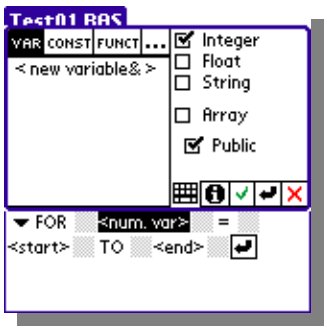


It has appeared now a new window for editing the whole line. The number of the line is shown in the title of this window: in this case ***Line 1***.
Here you can tap also the ***"i"*** button to get some help.
Tap the button ***Select a command***.

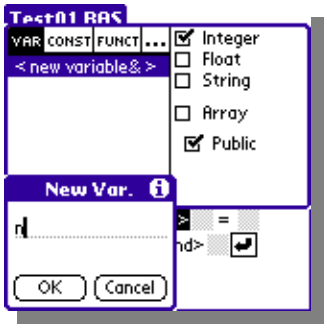
The first line will contain the sentence
FOR n = 0 TO 255
so tap the button ***F*** to go to first command which start with the letter ***F***, select the command ***FOR*** and then tap the button ***OK***. Another faster way to select a command is to double click the word (***FOR*** in this case) of the list.



Once the command has been selected, the required parameters are automatically added.
This is useful to check the syntax for each command.
Let's input the rest of the line.
Tap the button ***<num. var>***.

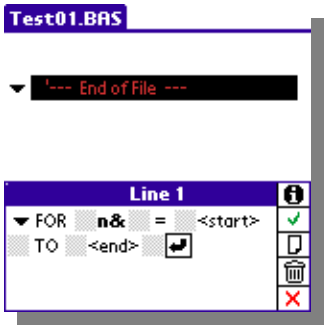


Now we have to create the integer variable ***n***.
Tap ***<new variable&>*** in the list and then the button with a green tick ✔ .
An alternative is to double click ***<new variable&>***.



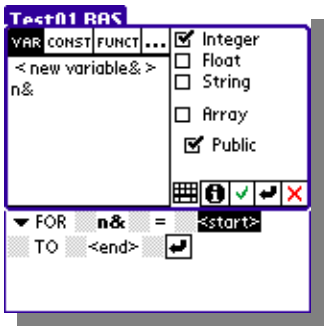
Enter the name of the variable.
In this case we will use ***n***.
Because it's an integer variable, it will have the suffix ***&*** but this mustn't be included in the name (it's automatically added).

Tap the button ***OK***.

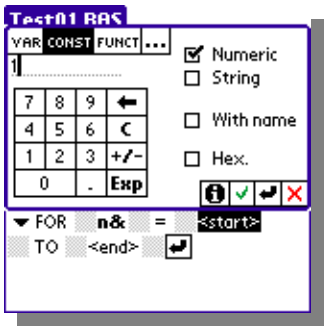


As you can see, the variable has been included in the line.

Now, we will input the ***<start>*** value.
Tap the word ***<start>***.

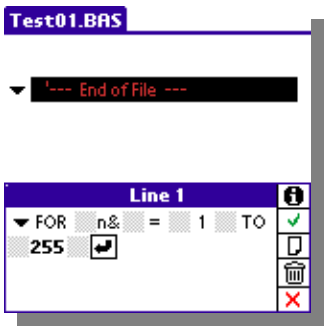


The value ***1*** is a constant, so tap the *push button* ***CONST***.



Input the value ***1***.
You can use the keypad which has appeared automatically.
Tap the green tick ✔.

Now, press the word ***<end>*** and input 255 in similar way than before.

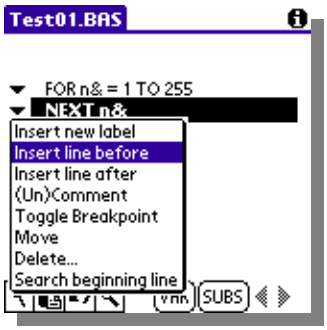


The first line is finished.
Now we can tap the green tick ✔.

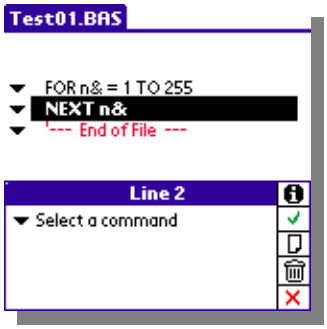
We will see for a short time a message saying ***AutoParenth: creating new line.***
This is because the command ***FOR*** must be programmed together with the command ***NEXT***.

And we can see that the corresponding line has been automatically created.

Now we want to insert a line between the FOR-line and the NEXT-line.



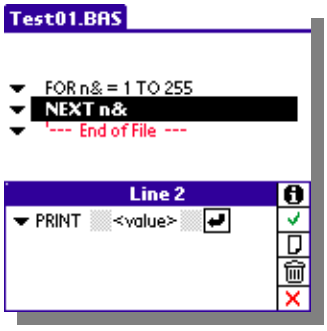
To do so, tap the triangle (pointing down) at the left of the NEXT-line and then tap ***Insert line before***.



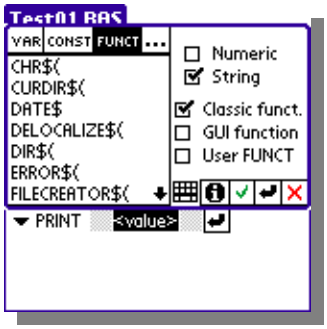
Now we can fill a line which will be inserted between both lines.
Note that the tile shows ***Line 2***.

This line must contain the sentence
PRINT CHR$(n&)

Select the command ***PRINT*** and then tap the button ***OK***.



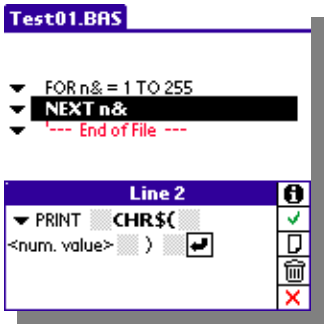
Tap the word ***<value>***.



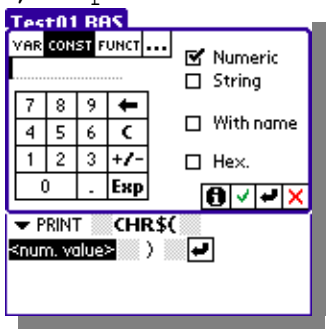
To input a string function, tap the *push button* ***FUNCT*** and then the *check box* ***String***.

Then select the function ***CHR$(*** and press the button ✔.

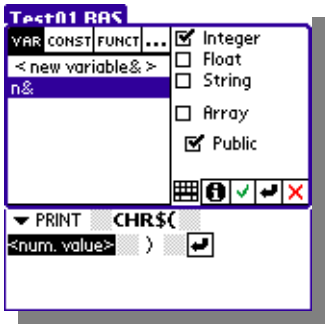
Alternatively, you can double click on ***CHR$(***.

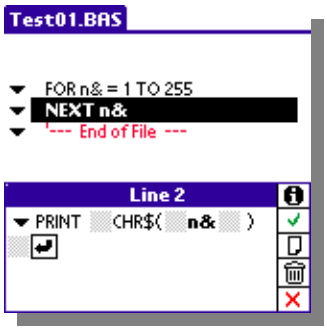


Now, tap the word ***<num. value>***.

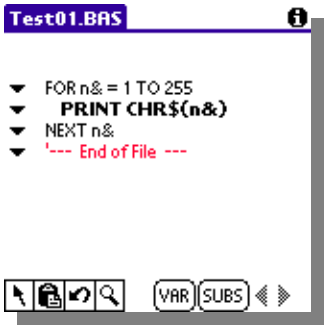


Now, tap the button ***VAR*** and select the variable ***n&***.





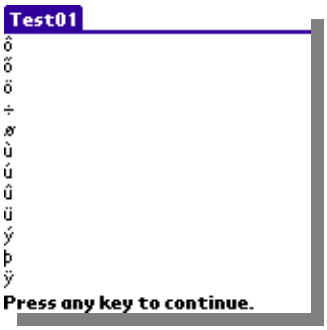
Now, the line number 2 is finished.
Tap the green ✔ to enter the line.



Our first program is finished.

Let's execute it.

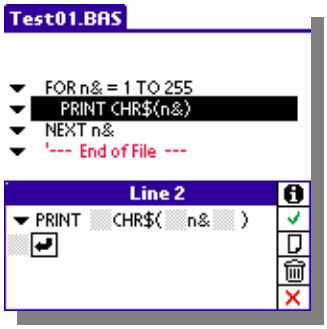
Tap the ***Menu*** button, select the group ***Run*** and then ***Run.***



Once the program has finished, appears the message ***Press any key to continue.***

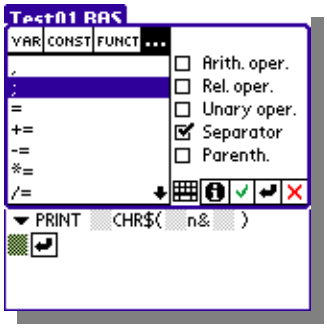
But we can't see most of the characters.

Let's modify the program.

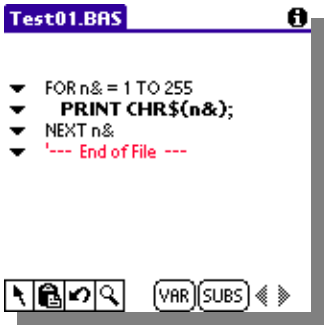


We will finish the 2nd line with a semicolon **";"** to print next character just after the previous one.

Tap the second line and then tap the grey space after the last parentheses and before the ENTER symbol ↵ .



Select the *check box* ***Separator*** and then select the semicolon ;
Tap the green ✓ to enter the line.

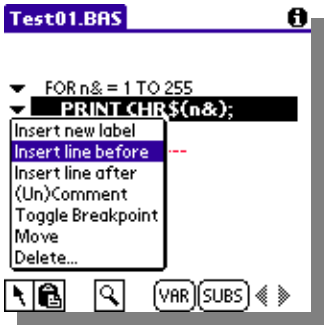


Our program is modified.
Let's execute again the program.

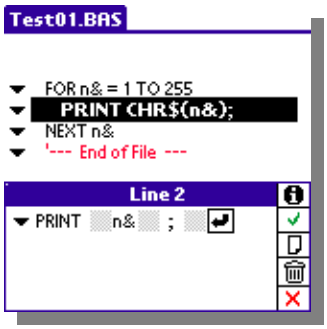
Ok, we can see all characters now.

But we can't know the number of each character.

So, let's modify again the program: we will print now the variable ***n&*** together with the ***CHR$(n&)***.

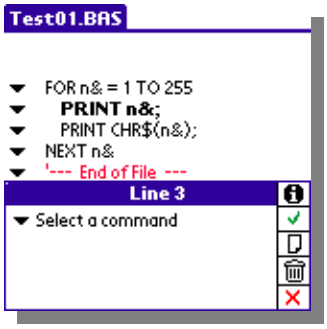


Insert a new line before the line that contains **PRINT CHR$(n&)**

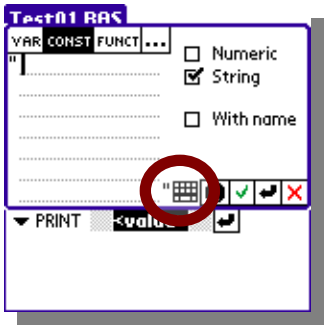


Create a line with this words:

PRINT n& ;



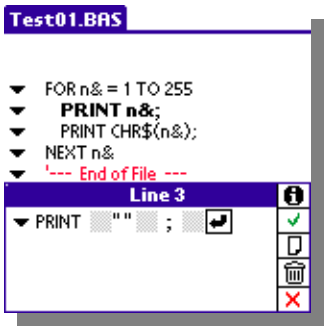
Tap the button ↵ instead of the green tick. This will open a new empty line just after this line.



Input a line containing these words:

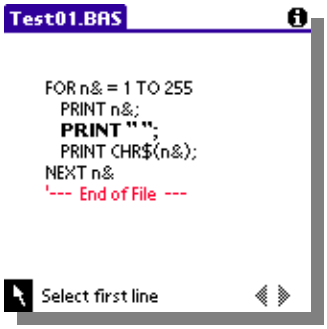
PRINT " " ;

To input a string constant, tap the button ***CONST*** and then the check box ***String***. If you need a keyboard on the screen, you can tap the icon which represents it (marked with a circle).



Tap the green ✔ to enter the line.

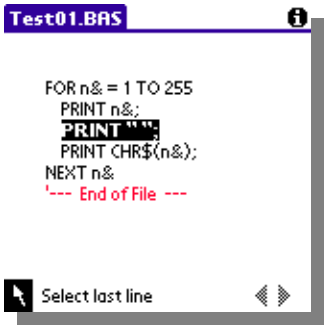
Now, we will copy the last line we have input.



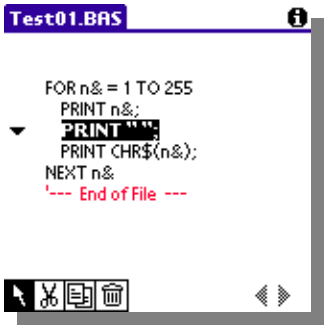
Tap the ***Select*** button, i.e. the one containing an arrow pointing to top-left.

At the bottom of the screen you can read ***Select first line***.

Tap the line we want to copy.



Now, we will tap the same line again, because the first line and the last line to select are the same.
If you want to cancel the selection mode, tap again the button with the arrow.



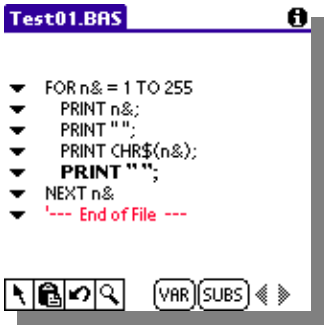
Once all the lines have been selected, the buttons for *Cut*, *Copy* and *delete* have appeared.

You can find some more options tapping the triangle at the left of the selected line.

Press the ***Copy*** button.

Then, press the ***Paste*** button and select where to paste the copied lines. The lines will be inserted *before* the line you select.
In this case, we want to copy it after *PRINT CHR$(n&);*

So select the line *NEXT n&*



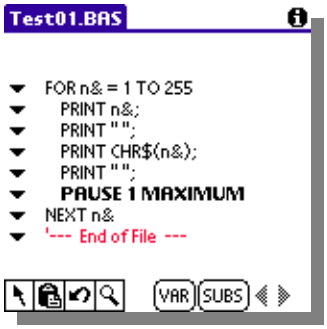
Now, we have made all the modifications.

Let's execute the program again.

Well. Let's analyse the result.

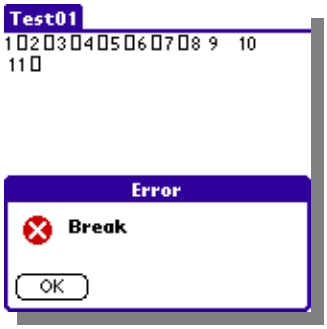
The screen has been scrolled and the first 181 characters have disappeared.

We will have to use a command to pause the execution.



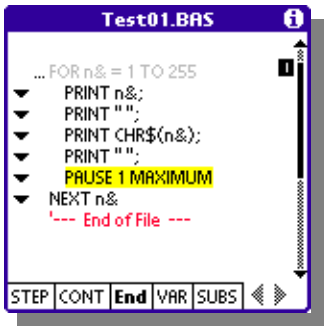
Insert before the line *NEXT N&* a new line with the words
PAUSE 1 MAXIMUM

and execute the program again.



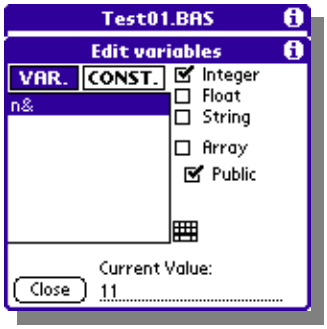
Oh! The program is running very slow!
Let's stop it: tap the *Applications Launcher* button (the one with a house).

We get an error screen which says ***Break***.
Tap the button ***OK*** to go to debugging screen.



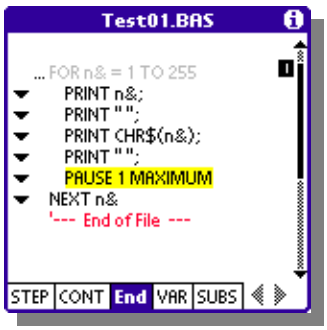
In this screen you can modify the lines, too.
But you can't create new lines nor delete them.

You can also check the value of any variable:
tap the button ***VAR***

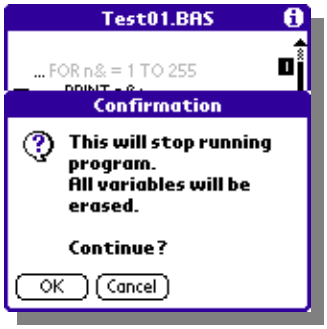


Select the variable you want to check.
Its current value appears at the bottom of the screen.

Close this screen.

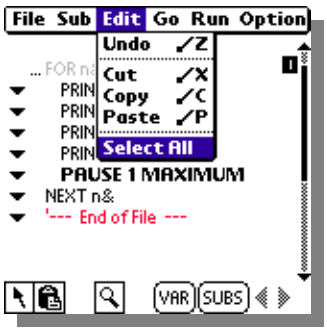


Stop the execution of the program, tapping the button ***End***.



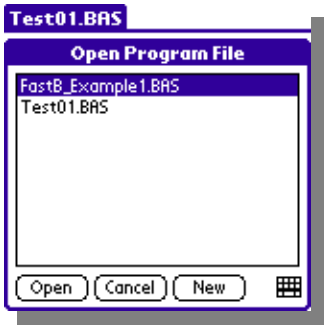
It will appear this message.

Tap the button ***OK*** to return to main screen.



We will delete all the line of program and start again.
To do so, open the menu, go to Edit and tap ***Select All***.

Now, press the button for deleting (it's the last one) and confirm the deletion.



Open the menu, go to File and tap ***Open program***.
Select the ***FastB_Example1.BAS***, which is provided with the examples.

We will copy all the program.
To do so, open the menu again, go to Edit and tap ***Select All***.
Now tap the button for Copy (or open the menu > Edit > Copy).

Open the program ***Test01.BAS*** again and press the paste button.

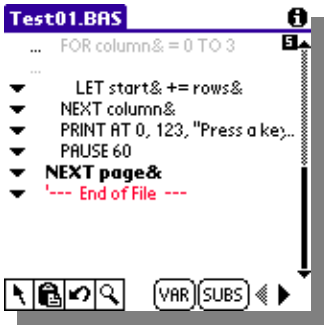
Select the line ***---End of File---*** (there is only this line to select.)

Now we get a message that says: The variable ***page*** doesn't exist.

We will create it automatically.
Select ***Create a new public variable*** and tap ***OK***.

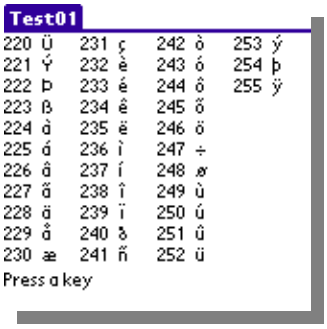
We will keep the name of the variable "page", so tap the button ***OK***.

Do the same with the rest of variables.



All the program has been pasted.

We will execute it to see the results



Well, this look better.

This is the end of **Quick Start**. You can check by yourself the rest of options.

Variables, Constants and Operators

The name of any variable must start by a letter, and can contain digits and underscore symbol (_).
The maximum length of the name is 15 characters.

Public variables can be used anywhere in the program, while private variable can be used only in the current sub.

Integer variable

The limits of this kind of variable are:
+2,147,483,647 for positive values.
-2,147,483,648 for negative values.

Each integer variable occupies 4 bytes of memory.

Floating point variable

The limits of this kind of variable are:
±1.79769E+308 for big values.
±2.22507E-308 for small values.

Each floating point variable occupies 8 bytes of memory.

String variable

One string can contain up to 63.5Kb (about 65000 characters), if there is enough free memory.

Each string variable occupies as many bytes of memory as the string length plus 4 bytes.

Arrays

The arrays can have up to 4 dimensions, and the maximum size for any array is 63.5Kb (65024 bytes).

The first element of each dimension can be 0 or 1, depending on value of 'Array base 1' in 'Program settings'.

The size of an array is calculated multiplying the number of elements by the size of the element (integer=4, float=8).
The size of STRING elements is 4 bytes. Then, each element is considered like a string variable and the limits are the same than the variables.
You must add then 16 bytes to total size, used for array data.

Arrays can be only Public, they can't be Private.

Named constants

You can create new named-constants and set their value in a program-line with command CONST.

The value of constants can't be modified during program execution, preventing possible errors.

A constant must be defined at the beginning of the program (before any executable command) and can be declared only in the Main Procedure.

Constants are Public, i.e. they can be used anywhere in the program.
The rules for the name are the same than for variables, and also the maximum and minimum values.

Numeric constants

When editing a line, if you want to input an integer constant, the limits are:
+2,147,483,647 for positive values.
-2,147,483,648 for negative values.

If you want to input a floating point constant, the limits are:
±1.79769E+308 for big values.
±2.22507E-308 for small values.

String constant

When editing a line, if you want to input a string constant, you can input up to 80 characters.

Operations

Addition
Syntax: expr1 _+_ expr2
    Add two expressions.
    If both expressions are strings, they are concatenated.

Subtraction
Syntax: expr1 _-_ expr2
    Returns the result of subtracting expr2 from expr1.
    Both expressions must be numeric.

Multiplication
Syntax: expr1 _*_ expr2
    Returns the result of multiplying two expressions.
    Both expressions must be numeric.

Float division
Syntax: expr1 _/_ expr2
    Returns the result of dividing expr1 by expr2.
    Both expressions must be numeric.
    Result is always in float mode. To execute a fast division
    between two integers, use integer division '\'.

Integer division
Syntax: expr1 __ expr2
    Returns the result of dividing expr1 by expr2.
    Both expressions must be numeric.
    Result is always in integer mode. If any of the expressions is
    not integer, it is converted to integer before making the
    division.

Power
Syntax: expr1 _^_ expr2
    Returns the result of raising expr1 to the power of expr2.
    Both expressions must be numeric.
    If any of both operands is a float number, then MathLib
    library is required.

Modulus
Syntax: expr1 _MOD_ expr2
    Returns the remainder of the division between expr1 and expr2.
    Both expressions must be numeric.
    If any of both operands is a float number, then MathLib
    library is required.

Comparing

- Syntax: expr1 _<_ expr2
  Returns -1 (true) if expr1 is less than expr2. Otherwise returns 0 (false).
  Expressions can be numeric or strings, but both must be of the same type. Float and integers can be mixed.
- Syntax: expr1 _<=_ expr2
  Returns -1 (true) if expr1 is less or equal to expr2. Otherwise returns 0 (false).
  Expressions can be numeric or strings, but both must be of the same type. Float and integers can be mixed.
- Syntax: expr1 _=_ expr2
  Returns -1 (true) if expr1 is equal to expr2. Otherwise returns 0 (false).
  Expressions can be numeric or strings, but both must be of

the same type. Float and integers can be mixed.

- Syntax: expr1 _>_ expr2
  Returns -1 (true) if expr1 is greater than expr2. Otherwise returns 0 (false).
  Expressions can be numeric or strings, but both must be of the same type. Float and integers can be mixed.
- Syntax: expr1 _>=_ expr2
  Returns -1 (true) if expr1 is greater or equal to expr2. Otherwise returns 0 (false).
  Expressions can be numeric or strings, but both must be of the same type. Float and integers can be mixed.
- Syntax: expr1 _<>_ expr2
  Returns -1 (true) if expr1 is different from expr2. Otherwise returns 0 (false).
  Expressions can be numeric or strings, but both must be of the same type. Float and integers can be mixed.
- Syntax: expr1 _AND_ expr2
  Perform a boolean bitwise AND between both expressions.
  ```
  0 AND 0 = 0
  0 AND 1 = 0
  1 AND 0 = 0
  1 AND 1 = 1
  ```
  For example:
  10 AND 6 gives a result of 2
  (BIN: 1010 AND 0110 = 0010)
- Syntax: expr1 _OR_ expr2
  Perform a boolean bitwise OR between both expressions.
  ```
  0 OR 0 = 0
  0 OR 1 = 1
  1 OR 0 = 1
  1 OR 1 = 1
  ```
  For example:
  10 OR 6 gives a result of 14
  (BIN: 1010 OR 0110 = 1110)
- Syntax: expr1 _XOR_ expr2
  Perform a boolean bitwise exclusive-OR between both expressions.
  ```
  0 XOR 0 = 0
  0 XOR 1 = 1
  1 XOR 0 = 1
  1 XOR 1 = 0
  ```
  For example:
  10 XOR 6 gives a result of 12
  (BIN: 1010 XOR 0110 = 1100)
- Syntax: _-_ numeric expression
  Changes the sign of expression.
- Syntax: _NOT_ numeric expression
  Returns the 2-complement of the number.

If the number is not an integer, it is converted to it. This operator changes the bits that are 0 to 1, and vice versa.
For example:

```
NOT 4 (=BIN 0000 0100)
   gives a result of
-5 (=BIN 1111 1011)
```

Derived Math functions

This is a list of math functions which can be derived from the available functions:

```
• arc-sine ASIN(x)
 = ATN(x / SQR(-x * x + 1))

• arc-cosine ACOS(x)
 = ATN(-x / SQR(-x * x + 1))
    + 2 * ATN(1)

• hyperbolic sine SINH(x)
 = (EXP(x) - EXP(-x)) / 2

• hyperbolic cosine COSH(x)
 = (EXP(x) + EXP(-x)) / 2

• hyperbolic tangent TANH(x)
 = (EXP(x) - EXP(-x))
    / (EXP(x) + EXP(-x))

• hyperbolic arc-sineASINH(x)
 = LOG(x + SQR(x * x + 1))

• hyperbolic arc-cosine ACOSH(x)
 = LOG(x + SQR(x * x - 1))

• hyperbolic arc-tang. ATANH(x)
 = LOG((1 + x) / (1 - x)) / 2

• base-n logarithm LOGn(x)
 = LOG(x) / LOG(n)
```

List of Commands: Normal Commands

Commands that assign a value to a variable

CONST

Syntax: CONST name = expression

Create a new named-constant.
Constants can't be defined more than once, and they must be declared at the beginning of the program (before any executable command) and only in the main procedure.

INPUT

Syntax: INPUT variable

Used to put a value into variable.

The system starts a line editor that finishes when user writes the ENTER key.
This line editor shows a blinking cursor on the screen, at the position determined by last PRINT command. You can modify this position programming
PRINT AT x, y, "";
before programming INPUT.

To stop the line editor, the user can press also on the 'Home'-silk button (used for opening the Applications Launcher). This action will generate a 'Break' error and stop the execution.

LET

Syntax: LET variable = expression

Stores the value of the expression into a variable.
If a float value is tried to be stored into an integer variable, it is first rounded to the nearest integer. For example: 1.3 is rounded to 1, and 1.7 is rounded to 2

Besides '=', some C-like separators can be used:

- += adds expression to variable (available also for strings).
- -= substracts expression from variable.
- *= multiplies variable by expression.
- /= divides variable into expression.
- ^= raises variable to the power of expression.

When using LET with arrays, the expression after the sign **'='** is always evaluated before variable.

For example:
```
     LET array(UserFn1) = array(UserFn2) + UserFn3(x)
```
The function *UserFn2* will be performed before *UserFn1*. To be exacts, the order will be: UserFn2, UserFn3, UserFn1.

LSET

Syntax: LSET variable$ = expression$

Stores the value of expression$ into a string variable. The data will be aligned to left.
LSET replaces the spare characters with spaces.
If the length of expression$ if larger than the current length of variable$ then LSET only copies the characters which are at the most left side.
To set the initial length of variable, you can use function SPC$, together with command LET:
```
    LET a$ = SPC$(Length%)
```

MID$

Syntax: MID$(var$, start, length) = value$

Replaces a specified number of characters in a string variable with characters from another string.

REDIM

Syntax: REDIM array (dimensions)

Erase the array and allocate new space for it.
The new array will be filled with zeroes.

The number of dimensions can't be changed. For example, if the array a$ has been dimensioned as a$(10,3)
then REDIM a$(20, 5) is ok,
but REDIM a$(20) is not valid, because it should be programmed with 2 dimensions.

RSET

Syntax: RSET variable$ = expression$

Stores the value of expression$ into a string variable. The data will be aligned to right.

RSET replaces the spare characters with spaces.
If the length of expression$ if larger than the current length of variable$ then RSET only copies the characters which are at the most right side.
To set the initial length of variable, you can use function SPC$, together with command LET:

```
LET a$ = SPC$(Length%)
```

SORT

Syntax: SORT array

Sorts an array from lower to higher. The number of dimensions must be 1.

SWAP

Syntax: SWAP variable1, variable2

Swaps the values of two variables.
Variables must be of the same type, i.e. float and integer variables can't be mixed.

Control Commands

CALL

Syntax: CALL UserSub

Transfers the control to a user Sub or Function.
It can be used also for calling a Function (which returns a value) when you want to discard the returned value.

CASE

The command CASE can have several forms:

Syntax 1: CASE expression
Used with SELECT CASE to evaluate a single expression.

Syntax 2: CASE IS-operator expression
Used with SELECT CASE to evaluate a expression with a relational operator.
Available operators are:

- CASE IS <= expression
- CASE IS < expression
- CASE IS = expression

- CASE IS > expression
- CASE IS >= expression
- CASE IS <> expression

Syntax 3: CASE start TO end
    Used with SELECT CASE to evaluate if a expression is within some specified limits.
    The smaller value must appear first. For example, the statements associated with the line CASE -1 TO -5 are not executed if the testexpresssion is -4. The line should be written as CASE -5 TO -1.

Syntax 4: CASE ELSE
    Used at the end of SELECT CASE group.
    The lines following CASE ELSE are executed only if the rest of CASE clauses have been false.

See command SELECT CASE for details.

DO

```
Syntax1:
DO WHILE | UNTIL condition
    statementblock
LOOP

Syntax2:
DO
    statementblock
LOOP WHILE | UNTIL condition
```

Repeat a group of instructions WHILE a condition is true, or UNTIL a condition becomes true.
With 'Syntax2', the loop is executed at least once, since the condition is not evaluated until the end.

DO…LOOP loops can be nested how many times you want.

You can use command EXIT DO to jump to the line after LOOP.

ELSE

Syntax: ELSE

See command IF … THEN for details.

END

Syntax: END

Stops program immediately.
Open files will be closed and variables will be erased.

In a compiled program, control returns to system.
In a non compiled one, a message is displayed and control returns to listing view after pressing any key.
If button 'break' is pressed instead of a key, variables are not erased and then you can check their value.

END IF

Syntax: END IF

See command IF … THEN for details.

END SELECT

Syntax: END SELECT

See command SELECT CASE for details.

ERROR

Syntax: ERROR num

Simulates the generation of an error.

EXIT DO

Syntax: EXIT DO

Used for exit a DO…LOOP loop. It transfers the control to the line following LOOP.

See command DO for details.

EXIT FOR

Syntax: EXIT FOR

Used for exit a FOR…NEXT loop. It transfers the control to the line following NEXT.

See command FOR for details.

EXIT SUB / FUNCTION

```
Syntax1: EXIT SUB
Syntax2: EXIT FUNCTION
```

Jumps to the end of a Sub or Function.

FOR

```
Syntax:
     FOR counter = start TO end [STEP step]
          …
          statements
          …
     NEXT counter
```

Repeat a group of statements for a specified number of times.

counter is a numeric variable (integer or float), used as a looping counter.
start is the initial value for the counter.
end is the final value for the counter.
The optional value step is added to the counter each time the loop is done. If it is omitted, the default value is 1.

If step is positive, the loop is executed if counter <= end. If step is negative, the loop is executed if counter >= end.
Once the loop is started and all statments in loop have been executed, the value step is added to counter. Then, the value of counter is checked again to see if loop must be repeated, or continue in the line following NEXT.

Modifying the value of counter while in the loop makes more difficult to read and debug the program.
The recommended way to exit from the loop before getting the end value is using EXIT FOR. It is used often in the checking of some condition (for example IF … THEN), and it transfers the control to the line following NEXT.

FOR … NEXT loops can be nested an unlimited number of times, but always using different counters.

GOSUB

```
Syntax: GOSUB label
```

Jumps to a subroutine (in the same procedure).

When the program finds a RETURN the control goes back to the line following GOSUB.

GOTO

Syntax: GOTO label

Jumps to the specified line-label.

IF … GOTO

Syntax: IF condition GOTO label

If condition is true, jumps to the specified line-label.
If not, continues to next line.

IF …THEN

Syntax:

```
IF condition THEN
     statementblockTrue
[ ELSE
   statementblockFalse ]
END IF
```

If condition is true, executes the statmentblockTrue.
If condition is false, jumps to statementblockFalse, if this is programmed.
If not programmed, jumps to the line following END IF.

NEXT

Syntax: NEXT counter

Used at the end of FOR … NEXT loops.

See command FOR for details.

ON ERROR

Syntax1: ON ERROR GOTO label
Syntax2: ON ERROR GOTO 0

Enables an error-handling routine and specifies the location of the routine within a procedure.

If a line-label is specified, the error-handling routine that starts at the line specified is enabled. If a run-time error

occurs, control branches to the specified line, making the error handler active. The specified line must be in the same procedure as the ON ERROR statement.

If you set the label to 0, the error-handling is disabled, so any run-time error that occurs is fatal: an error message is displayed, and execution stops.

If an error occurs while an error handler is active (before the RESUME command), the current error handler cannot handle the error and an error message is displayed, and execution stops.

RESUME

Syntax1: RESUME label
Syntax2: RESUME NEXT

Resumes execution after an error-handling routine is finished.

If a line-label is specified, execution resumes with that line. If you specify NEXT, execution resumes with the statement immediately following the statement that generated the error.

RETURN

Syntax: RETURN

Returns from a subroutine called with GOSUB.

RUN

Syntax: RUN fileName

Open a program and execute it. The file must be in RAM memory. If the file is in any external card, an error will occur.
In non-compiled mode, the expected type of file is a Basic file. Termination ".BAS" must not be added. It will be automatically added.
In compiled mode, it will open any application.

SELECT CASE

Syntax:

```
SELECT CASE testexpression
     CASE expression1
          [statementblock-1]
     CASE expression2
          [statementblock-2]
```

```
    …
         [CASE ELSE]
              [statementblock-n]
    END SELECT
```

testexpression is any numeric or string expression

statementblock… consist of any number of statements of one or more lines.

expression1 and expression2: These elements can have any of the three following forms:

- CASE expression
- CASE expr1 TO expr2
- CASE IS expression

expression is any numeric or string expression. The type of the expression must match the type of the testexpression. Integer and float expressions can be mixed. They are automatically converted.

If the testexpression matches the expression associated with a CASE clause, then the statement block following that CASE clause is executed up to the next CASE clause or, for the last one, up to END SELECT. Control then passes to the statement following END SELECT.

If you use the TO keyword to indicate a range of values, the smaller value must appear first. For example, the statements associated with the line CASE -1 TO -5 are not executed if the testexpresssion is -4. The line should be written as CASE -5 TO -1.

You may use a relational operator only with CASE_is command and the IS keyword.

If CASE ELSE is used, its associated statements are executed only if the testexpression does not match any of the other CASE selections. It is a good idea to have a CASE ELSE statement in your SELECT CASE block to handle unforeseen testexpression values.

When there is no CASE ELSE statement, and no expression listed in the CASE clauses matches testexpression, program execution continues normally. No error occurs.

You may use multiple expressions or ranges in each CASE clause, but you must input each CASE command in several lines. For example, the following lines are valid:

```
    CASE 1 TO 4
    (…CASE) 7 TO 9
```

```
     (…CASE) 11
     (…CASE) 13
     (…CASE) IS > MaxNumber%
```

If an expression appears in more than one CASE clause, only the statements associated with the first appearance of the expression are executed.

SELECT CASE statements may be nested. Each SELECT CASE statement must have a matching END SELECT statement.

STOP

Syntax: STOP

Stops program and goes to debugging view.

In a compiled program, a 'Break' error is generated and control returns to system.

WEND

Syntax: WEND

Ending part of a WHILE … WEND loop.

See command WHILE for details.

WHILE

Syntax:

```
     WHILE condition
          …
          statementblock
          …
     WEND
```

Executes statementblock while condition is true.
If condition is false at the first time, statementblock is not executed.

Command DO … LOOP offers a better functionality than WHILE … WEND.
See command DO for details.

Commands that draw on the screen

ARC

```
Syntax: ARC x1, y1 TO x2, y2 RADIUS r
```

Draws an arc in screen, from one point to the other.
The centre is automatically calculated.

If you want to draw in different colours, command INK must be programmed before.
You can also draw in inverted mode programming command INVERT before.

CIRCLE

```
Syntax: CIRCLE x, y RADIUS r [FILLED]
```

Draws a circle in screen, with centre at (x,y).
If the optional word FILLED is used, then the interior of the circle is painted with foreground colour.

If you want to draw in different colours, command INK must be programmed before.
You can also draw in inverted mode programming command INVERT before.

CLS

```
Syntax: CLS
```

Clears the screen and set the current position for printing and drawing to the top-left corner.

LINE

```
Syntax: LINE x1, y1 TO x2, y2
```

Draw a line from (x1,y1) to (x2,y2).

If you want to draw in different colours, command INK must be programmed before.
You can also draw in inverted mode programming command INVERT before.

LINE TO

```
Syntax: LINE TO x2, y2
```

Draw a line from last drawn point to (x2,y2).

See command LINE for details.

POINT

Syntax: POINT x, y

Draw a pixel using the current foreground colour.

See command LINE for details.

PRINT

Syntax: PRINT expression

Shows expression on the screen. If expression does not fit within the screen, it is truncated in several lines.

You can use at the end of the line some separators to define behavior after expression is shown.
With no separator, next PRINT will write on next line.
With semicolon (;) separator, next PRINT will write immediately after last shown character.
With comma (,) separator, next PRINT will write at a distance of one tabulator space.

If you want to PRINT in different colours, or with different fonts, commands INK or FONT must be programmed before.
You can also draw in inverted mode programming command INVERT before.

PRINT AT

Syntax: PRINT AT x, y, expression

Shows expression on the screen, starting at the position (x, y).

This command only shows the characters that fit within the screen, contrary to command PRINT, which truncates the expression in several lines.

See command PRINT for details about end-separator.

You can use this command for positioning printing cursor, for example:
PRINT AT x, y, “”;

RECTANGLE

Syntax: RECTANGLE x1, y1 TO x2, y2 [FILLED]

Draw a box with the points (x1,y1) and (x2,y2) specifying diagonally opposite corners.
If the optional word FILLED is used, then the interior of the box is painted with foreground colour.

If you want to draw in different colours, command INK must be programmed before.
You can also draw in inverted mode programming command INVERT before.

File Commands

CHDIR

Syntax: CHDIR DirName

Changes the current directory in an external card.
This command is not valid for RAM storage.

CHDRIVE

Syntax: CHDRIVE DriveName

Changes the current drive.

CLOSE

Syntax: CLOSE #FileNum

Close an open file.

FILE COPY

Syntax: FILE COPY source TO destination

Copies a file.
The file names can include the directory and the drive.

When copying from Ram to an external card, the termination .pdb or .prc is added automatically.

GET

Syntax: GET #FileNum, variable$

Read data from an open file, and set these data to a string variable. The file must have been opened in RANDOM, BINARY or INPUT mode.
In RANDOM mode, the working unit is a record, while in BINARY mode it is a byte.
Data read with GET # are usually written with PUT #.

These are the rules for files open in **RANDOM** mode:
The record number must be selected before with command SEEK #. If SEEK is not used, the reading record is the one following last GET # or PUT #.
The full content of the record is passed to string. PalmOS Devices are based in C language. This means that strings written with other Palm applications (for example MemoPad) can have a CHR$(0) at the end.

These are the rules for files open in INPUT or BINARY mode:
The position where to read must be selected before with command SEEK #. If SEEK is not used, the reading position is the one following last GET # or PUT #.
The length of string must be the same as the number of bytes that must be read. For example, to read 10 bytes from current position:

```
LET a$ = STRING(10, " ")
GET #1, a$
```

GET RSRC

Syntax: GET RSRC #FileNum, RsrcType, RsrcID, variable$

Read a resource record from an open file, and set the data to a string variable. The file must have been opened in RANDOM RESOURCE mode.
If the specified resource doesn't exists, the variable is set to empty string "".

INPUT

Syntax: INPUT #FileNum, variable

Read data from a sequential open file, and set these data to a variable. The file must have been opened in INPUT or BINARY mode.
When reading a number, it must be written using the dot '.' as decimal separator.
Data read with INPUT # are usually written with WRITE #.

KILL

Syntax: KILL FileName

Erases permanently a file.
The file name can include the directory and the drive.

LINE INPUT

Syntax: LINE INPUT #FileNum, string variable

Read a full line from a sequential open file, and set this line to a string variable. The file must have been opened in INPUT or BINARY mode.
Data read with LINE INPUT # are usually written with PRINT #.

MKDIR

Syntax: MKDIR DirName

Creates a new directory in an external card.
This command is not valid for RAM storage.

NAME

Syntax: NAME OldName AS NewName

Renames a file.
old name can include the directory and the drive, but new name can't.

OPEN

Syntax: OPEN name FOR mode AS FileNum [CREATOR creator TYPE type]

Open a file to enable input/output operations.

- name is a string which contains a valid file name. Directory and card unit can be included, like in a PC. See commands CHDRIVE and CHDIR for details.
- mode specifies how to open the file. Valid modes are INPUT, OUTPUT, APPEND, BINARY to work with sequential files, or RANDOM or RANDOM_RESOURCE to work with databases.
  - INPUT for sequential reading (with INPUT# or> LINE INPUT#If file doesn't exist, an error occurs.
  - OUTPUT for sequential writing (with PRINT# or WRITE#). If file already exists, it is deleted first. If file doesn't exist, it is created.

    - APPEND for sequential writing. If file already exists, new data are added after existing data. If file doesn't exist, it is created.
    - BINARY for sequential reading or writing with GET# or PUT#. Written or read data are a set of bytes, whose length depends on a string-variable length. If file doesn't exist, it is created.
    - RANDOM for database reading or writing with GET# or PUT#. Valid only for databases in RAM. To open a database in an external card, use BINARY mode. If file doesn't exist, it is created.
    - RANDOM_RESOURCE for PRC-database reading or writing with GET RSRC# or PUT RSRC#. If file doesn't exist, it is created.
- FileNum is an integer value between 1 and 255.
- Files which are stored in RAM have a creator and type attributes. These are a 4-chars strings (see Palm Documentation for details). The following ones are the default values if they are not specified (recommended):
    - For sequential files, creator ''REAd'' and type ''TEXt'' (PalmDoc files).
    - For random files, creator will be the same than current application (see menu Options > This program settings) and type will be ''DATA''.

To open a sequential file that is not a Palm Doc file, the Palm OS version must be 3.0 or greater. You should use SYSVAR (OSVERSION) and check that the result is >= &H0300

PRINT

Syntax: PRINT #FileNum, expression

Write expression to a sequential file. The file must have been opened in OUTPUT, APPEND or BINARY mode.
The same separators used with PRINT can be used here too.
The data written with PRINT # are usually read with LINE INPUT #

PUT

Syntax: PUT #FileNum, string [NEW]

Write the data contained in string to an open file. The file must have been opened in RANDOM, BINARY, OUTPUT or APPEND mode.
In RANDOM mode, the working unit is a record, while in BINARY mode it is a byte.
The data written with PUT # are usually read with GET #.

These are the rules for files open in **RANDOM** mode:
The record number must be selected before with command SEEK #. If SEEK is not used, the reading record is the one following last GET # or PUT #.
The optional word NEW will create a new record (only available in RANDOM mode). If word NEW is omitted, then the existing record is replaced with the passed string.
To erase a record, pass an empty string "" (only RANDOM mode).
PalmOS Devices are based in C language. This means that strings which can be used for other Palm applications (for example MemoPad) must have a CHR$(0) at the end.

These are the rules for files open in **BINARY**, **OUTPUT** or **APPEND** modes:
The position where to write must be selected before with command SEEK #. If SEEK is not used, the writing position is the one following last GET # or PUT #.
The optional word NEW is not allowed. Data are written in a sequential way. If file is not large enough, it is enlarged.

PUT RSRC

Syntax: PUT RSRC #FileNum, RsrcType, RsrcID, string

Write the data contained in string to a open file. The file must have been opened in RANDOM RESOURCE mode.
If the specified resource already exists, then the existing record is replaced with the passed string.
To erase a record, pass an empty string "".

RMDIR

Syntax: RMDIR DirName

Removes a directory in an external card.
The directory must be empty before removing it. You can check if there is any file in the dir using the function NUMFILES().
This command is not valid for RAM storage.

SEEK

Syntax: SEEK #FileNum, position

Set the position for the next reading or writing operation of an open file.
This instruction is not valid for files open in APPEND or RANDOM RESOURCE modes.

WRITE

Syntax: WRITE #FileNum, expression

Write expression to a sequential file. The file must have been opened in OUTPUT, APPEND or BINARY mode.
For float numbers, it uses always the dot '.' as decimal separator, regardless of local format, contrary to PRINT #, which uses local format.
When writing a string, quotation marks are also written at the beginning and at the end.
The data written with WRITE # are usually read with INPUT #

Other Commands

BEEP

Syntax: BEEP number

Play a pre-defined (simple) system sound.
The available beep numbers are:

- 1: **SNDINFO** Heralds non-crucial information.
- 2: **SNDWARNING** Grabs the user's attention.
- 3: **SNDERROR** Indicates an illegal operation.
- 4: **SNDSTARTUP** Played at device start up time.
- 5: **SNDALARM** Generic alarm sound; note that this is not the Datebook's alarm sound.
- 6: **SNDCONFIRMATION** Indicates approval or acceptance.
- 7: **SNDCLICK** The button click sound.

PAUSE

Syntax: PAUSE seconds [MAXIMUM]

Waits the number of seconds specified or, if MAXIMUM is specified, until an event occurs (for example, a key is pressed).
This means that if MAXIMUM is specified the pause-time can be shorter than the specified time. If MAXIMUM is not written, the elapsed time is exactly the programmed time.

If seconds is zero, then waits forever, until a key is pressed or a button is tapped. The word MAXIMUM is ignored.

RANDOMIZE

Syntax: RANDOMIZE seed

Initialize the ramdom-numbers generator of RND function and set seed as the seed number for it.

If RANDOMIZE is never used, the function RND uses as seed number always the same. It means that the values returned by RND are the same every time you start a program.

If seed is zero, it isn't modified.

Usually this command is programmed in this way:
RANDOMIZE TIMER

REM

Syntax: REM remark

This command is used only to include explanatory remarks in a program.

SEND KEYS

Syntax: SEND KEYS string

Send one or more keystrokes like if they were pressed on the keyboard.

SEND CHAR

Syntax: SEND CHAR value

Send one character as if a key was pressed on the keyboard.

SOUND

Syntax: SOUND frequency, milliseconds, volume

Perform a simple sound synthesis operation.

List of Commands: GUI Commands

Commands that create objects

NEW BITMAP

```
Syntax: NEW BITMAP #objectID, bitmapID AT x, y

Create a new bitmap in the current form.
Palm OS 3.0 or greater is needed to use this command.

The bitmapID is a numeric value identifying the resource that
provides the bitmap. This value must be unique within the
application scope.
Usually the objectID and the bitmapID are the same, but this is
not mandatory.
```

NEW BUTTON

```
Syntax:
NEW BUTTON #objectID, title$ AT x, y WIDTH width HEIGHT height

Create a new control object dynamically and install it in the
current form.
Palm OS 3.0 or greater is needed to use this command.

The style of the control will be defined as a button.
Font to be used must be specified before with command FONT.
```

NEW CHECKBOX

```
Syntax:
    NEW CHECKBOX #objectID GROUP group, title$ AT x, y WIDTH width
    HEIGHT height

Create a new control object dynamically and install it in the
current form.
Palm OS 3.0 or greater is needed to use this command.

The style of the control will be defined as a check box.
Font to be used must be specified before with command FONT.
```

NEW FORM

```
Syntax:
NEW FORM #formID, title$ AT x, y WIDTH width HEIGHT height [CLEAN]
```

Create a new form dynamically.
Palm OS 3.0 or greater is needed to use this command.

If CLEAN is used, screen is cleaned before drawing the new form, and form will be not modal.
If not used, form will be modal.

To remove the created form, use the command CLOSE FORM

NEW LABEL

Syntax: NEW LABEL #objectID, title$ AT x, y

Create a new label object dynamically and install it in the current form.
Palm OS 3.0 or greater is needed to use this command.

Font to be used must be specified before with command FONT.

NEW LIST

Syntax:

```
NEW LIST #objectID AT x, y WIDTH width HEIGHT height OPTIONS
array$
```

Create a new list object dynamically and install it in the current form.
Palm OS 3.2 or greater is needed to use this command.

Font to be used must be specified before with command FONT.

NEW NUMFIELD

Syntax:

```
NEW NUMFIELD #objectID MAXLEN MaxChars AT x, y WIDTH width
HEIGHT height
```

Create a new field object dynamically and install it in the current form.
Palm OS 3.0 or greater is needed to use this command.

The style of the field will be defined as numeric.
Font to be used must be specified before with command FONT.

NEW POPUPLIST

Syntax:

```
NEW POPUPLIST #objectID AT x, y WIDTH width HEIGHT height
```

```
    OPTIONS array$
```

Create a new list object dynamically and install it in the current form.
Palm OS 3.2 or greater is needed to use this command.

The style of the list will be defined as a popup-list, so it will not be displayed on the screen until function POPUPLIST is called.
Font to be used must be specified before with command FONT.

NEW POPUPTRIGGER

Syntax:

```
    NEW POPUPTRIGGER #objectID title$ AT x, y WIDTH width HEIGHT
    height OPTIONS array$ LISTROWS list rows
```

Create a new control object dynamically and install it in the current form.
Palm OS 3.2 or greater is needed to use this command.

The style of the control will be defined as a popup trigger.
Font to be used must be specified before with command FONT.

NEW PUSHBUTTON

Syntax:

```
    NEW PUSHBUTTON #objectID GROUP group, title$ AT x, y WIDTH
    width HEIGHT height
```

Create a new control object dynamically and install it in the current form.
Palm OS 3.0 or greater is needed to use this command.

The style of the control will be defined as a push button.
Font to be used must be specified before with command FONT.

NEW RPTBUTTON

Syntax:

```
NEW RPTBUTTON #objectID, title$ AT x, y WIDTH width HEIGHT height
```

Create a new control object dynamically and install it in the current form.
Palm OS 3.0 or greater is needed to use this command.

The style of the control will be defined as a repeating button.
Font to be used must be specified before with command FONT.

NEW SELECTTRIGGER

```
Syntax:
    NEW SELECTTRIGGER #objectID title$ AT x, y WIDTH width HEIGHT
    height OPTIONS array$ LISTROWS list rows

Create a new control object dynamically and install it in the
current form.
Palm OS 3.2 or greater is needed to use this command.

The style of the control will be defined as a selector trigger.
Font to be used must be specified before with command FONT.
```

NEW SLIDER

```
Syntax:
    NEW SLIDER #objectID AT x, y WIDTH width HEIGHT height FROM
    MinValue TO MaxValue STEP PageSize

Create a new slider object dynamically and install it in the
current form.
Palm OS 3.5 or greater is needed to use this command.

The initial value will be set to zero. To set other values, use
the commands VALUE OBJECT and LIMITS OBJECT.
```

NEW STRFIELD

```
Syntax:
    NEW STRFIELD #objectID MAXLEN MaxChars AT x, y WIDTH width
    HEIGHT height

Create a new field object dynamically and install it in the
current form.
Palm OS 3.0 or greater is needed to use this command.

The style of the field will be defined as alphanumeric.
Font to be used must be specified before with command FONT.
```

Commands that modify objects

DISABLE OBJECT

```
Syntax: DISABLE OBJECT #objectID

Set a control as disabled. Disabled controls do not respond to the
pen.
The control is not hidden, it is still shown.
```

ENABLE OBJECT

Syntax: ENABLE OBJECT #objectID

Set a control as enabled. Disabled controls do not respond to the pen.
The control is not redrawn.

FOCUS OBJECT

Syntax: FOCUS OBJECT #objectID

Set the focus of a form to the specified object. The object must be a field.

FRM TITLE

Syntax: FRM TITLE title$

Change the title of current form.

HIDE OBJECT

Syntax: HIDE OBJECT #objectID

Hide the specified object and set its attribute data (usable bit) so that it does not redraw or respond to the pen.

LIMITS OBJECT

Syntax: LIMITS OBJECT #objectID FROM MinValue TO MaxValue

Change the limits of the specified object. The object must be a scroll bar or a slider.

MOVE OBJECT

Syntax: MOVE OBJECT #objectID TO x, y

Move an object to the specified position.

OPTIONS LIST

Syntax: OPTIONS LIST #objectID = array [REDRAW]

Change the options for a list.

```
If REDRAW is not specified, then the display is not updated.
```

OPTIONS TRIGGER

```
Syntax:
OPTIONS TRIGGER #objectID = array

Change the options for a trigger.
The display is not updated.
```

REMOVE OBJECT

```
Syntax: REMOVE OBJECT #objectID

Remove an object created dynamically.
```

RESIZE FORM

```
Syntax: RESIZE FORM x, y WIDTH width HEIGHT height

Resize and move the current form.
```

RESIZE OBJECT

```
Syntax: RESIZE OBJECT #objectID TO width, height

Resize an object.
```

SELECT FIELD

```
Syntax: SELECT FIELD #objectID FROM start TO end

Set the current selection in a field and highlight the selection.
To cancel a selection, set both startPosition and endPosition to
the same value.
```

SELECT GROUP

```
Syntax: SELECT GROUP #group = objectID

Set the selected control in a group of controls.
This command unsets all the other controls in the group. The
display is updated.
```

SELECT LIST

```
Syntax: SELECT LIST #objectID = position
```

Set the selection for a list.
The old selection, if any, is unselected. If the list is visible, the selected item is visually updated. The list is scrolled to the selection, if necessary, as long as the list object is both visible and usable.

The first item can be 1 or 0, depending on value of the field Array base 1 in the menu option This program settings, it is, the value returned by LBOUND().

SHOW OBJECT

Syntax:
SHOW OBJECT #objectID

Set an object as usable and draw it.

TEXT OBJECT

Syntax: TEXT OBJECT #objectID = title

Change the text of the specified object. The object must be a field or a control.
If the object is a numeric field, the string must contain also a numeric value.

TOPITEM

Syntax: TOPITEM LIST #objectID = position [REDRAW]

Set the item visible. The item cannot become the top item if it's on the last page.
The value you specify for itemNum must be in the range 0 to max-number-of-items. If REDRAW is not specified, then the display is not updated.

The first item can be 1 or 0, depending on value of the field Array base 1 in the menu option This program settings, it is, the value returned by LBOUND().

VALUE OBJECT

Syntax: VALUE OBJECT #objectID = value

Change the value of the specified object. The object must be a scroll bar, a slider, a check box or a push button.
If the object is a check box or a push button, the value means 0

for unselected or 1 for selected.

Commands that affect the screen

CLOSE FORM

Syntax: CLOSE FORM

Close the current form.

COORDINATES

Syntax: COORDINATES XYzero

Modify the XY-zero of the screen, for subsequent commands like PRINT AT, LINE, CIRCLE…

Available values for XYzero are:

- **TOP LEFT**: Sets the origin of X and Y at the top left corner. The Y-axis increases downwards. This is the default mode.
- **MIDDLE CENTER**: Sets the origin of X and Y in the middle of the screen. The Y-axis increases upwards.
- **BOTTOM LEFT**: Sets the origin of X and Y at the bottom left corner. The Y-axis increases upwards.

FONT

Syntax: FONT number

Change the font for subsequent instructions (PRINT, NEW BUTTON …)

The available font numbers are:

- 0: **Standard** plain text font. A small standard font used to display user input. This font is small to display as much text as possible.
- 1: **Bold** font. Same size as STANDARD font but bold for easier reading. Used for text labels in the user interface.
- 2: **Large** font. A larger font provided as an alternative to STANDARD font for users who find the standard font too small to read.
- 3: **Symbol** font. Contains many special characters such as arrows, shift indicators, and so on.
- 4: **Symbol11** font. Contains the check boxes, the large left arrow, and the large right arrow.
- 5: **Symbol7** font. Contains the up and down arrows used for the repeating button scroll arrows and the dimmed version of the same arrows.

- 6: **Led** font. Contains the numbers 0 through 9, –, ., and the comma (,). Used by the Calculator application for its numeric display.
- 7: **Large bold** font (OS version 3.0 or later).

To set the default system font, set a value of -1.
If a number different to these is tried to be set, an error is generated.

FONT USER

Syntax: FONT USER rsrcID

Change the font to a custom font for subsequent instructions (PRINT, NEW BUTTON …)

Palm OS 3.5 or greater is needed to use this command.

The font must be defined in the 'Included Rsrc file' with a Rsrc-type of 'nfnt' or 'NFNT'.

FRM REDRAW

Syntax: FRM REDRAW

Clears the screen and then redraw all form objects (buttons, …). The current position for printing and drawing is set to the top-left corner.

HIRES SCALING

Syntax: HIRES SCALING ON | OFF

Specifies whether bitmaps and fonts should be scaled.
If you need to display a large, low-density bitmap (for instance, a map or a photograph) you can use this command to draw your bitmap unscaled, allowing the user to see more of the bitmap at one time.
If you want to draw more, but smaller, text on a handheld that has a high-density display (including 1.5 and double-density), you can use this command to draw text unscaled.
On a handheld with a high-density display, if the bitmap being drawn, or the font being used, is part of a family that contains both low- and high-density members, this command controls which member is used. If Scaling is set to OFF, for instance, the low-density font is used, unscaled, when drawing subsequent text.

INK

Syntax: INK foreground

Change the foreground colour for subsequent drawing instructions (PRINT, LINE…)

To calculate the value to pass, use the function RGB( )
To set the default system colour, set a value of -1.

Palm OS 3.5 or greater is needed to use this command.
In lower OS, this command does nothing.

To see the available colours (in a device with colour screen), select 'Available colours' in 'See also' list.

INVERT

Syntax: INVERT ON | OFF

When ON, swap foreground and background colours for subsequent drawing instructions (PRINT, LINE…)

OPEN FORM

Syntax: OPEN FORM #formID [CLEAN]

Open the specified form. You must have included a resources file.

If CLEAN is used, screen is cleaned before drawing the new form, and form will be not modal.
If not used, form will be modal.

PAPER

Syntax: PAPER background

Change the background colour for subsequent drawing instructions (PRINT, LINE…)

To calculate the value to pass, use the function RGB( )
To set the default system colour, set a value of -1.

Palm OS 3.5 or greater is needed to use this command.
In lower OS, this command does nothing.

To see the available colours (in a device with colour screen), select 'Available colours' in 'See also' list.

RESOLUTION

Syntax: RESOLUTION density

Changes the resolution of the screen.

Available values for density are:

- **STANDARD DENSITY:** This is the default mode. This mode is available in all devices. The usual size of the screen in this mode is 160x160 pixels.
- **ONE AND A HALF:** Not all devices support this mode. If this density is not supported, the system will set the STANDARD density.
- **DOUBLE:** This is the Hi-Res mode. Not all devices support this mode. The usual size of the screen in this mode is 320x320 pixels. If this density is not supported, the system will set the STANDARD density.
- **TRIPLE:** Not all devices support this mode. If this density is not supported, the system will try to set the DOUBLE density. If this is not supported too, then will set the STANDARD density.
- **QUADRUPLE:** Not all devices support this mode. If this density is not supported, the system will try to set the DOUBLE density. If this is not supported too, then will set the STANDARD density.

If the programmed density is not available, the system will set the maximum supported density, lower than the programmed one.

Other GUI-Commands

ALERT

Syntax: ALERT #alertID, text1$, text2$, text3$

Create a modal dialogue from an alert resource and display the dialogue until the user taps a button in the alert dialogue.

Up to three strings can be passed to this routine. They are used to replace the variables ^1, ^2 and ^3 that are contained in the message string of the alert resource.

DRAW BITMAP

Syntax: DRAW BITMAP #bitmapID AT x, y

Draws the specified bitmap in the current form.

The bitmapID is a numeric value identifying the resource that provides the bitmap. This value must be unique within the application scope.

MSGBOX

Syntax: MSGBOX prompt, [buttons, title]

Displays a message in a dialog box and waits for the user to click a button

- prompt is a string expression displayed as the message in the dialog box.
- buttons is an optional numeric expression specifying the number and type of buttons to display.
  - OK ONLY: Displays only the OK button.
  - OK CANCEL: Displays OK and Cancel buttons.
  - ABORT RETRY IGNORE: Displays Abort, Retry and Ignore buttons.
  - YES NO CANCEL: Displays Yes, No and Cancel buttons.
  - YES NO: Displays Yes and No buttons.
  - RETRY CANCEL: Displays Retry and Cancel buttons.

  If omitted, the default value is OK ONLY.
- title is an optional string displayed in the title bar of the dialog box.

WAIT

Syntax: WAIT ticks [LOWLEVEL]

Wait for an event.

ticks is the maximum number of ticks to wait before an event is returned (-1 means wait indefinitely).
If the separator LOWLEVEL is NOT specified, only the high-level events (keydown, button pressed…) are taken into account.
If specified, when any kind of event occurs, the control returns to next line.
If LOWLEVEL is used, you should also use MODE VERY FAST to get a correct behaviour.
To get the event that occurred, use function LASTEVENT.

List of Commands: Extended Commands

MODE

```
Syntax1: MODE NORMAL
Syntax2: MODE FAST
Syntax3: MODE VERY FAST
```

Set the running mode.
In compiled programs this instuction is ignored, because it's always running in very fast mode.

In FAST mode, the execution becomes faster. The system checks the pressed keys only in lines that have a jump backwards (NEXT, WEND…)
If the program comes into a endless loop, the only way to stop the program is tapping the 'Home'-silk button.
In VERY FAST mode, there is no way to stop the program, so be very careful when using it. It should be used only together with WAIT LOWLEVEL

PLAY MIDI

```
Syntax: PLAY MIDI string$, volume
```

Plays a midi string.
Palm OS 3.0 or greater is needed to use this command.
In Palm OS, the string needs a special header. If it doesn't exist, you should insert the following string at the beginning:
“PMrc” + CHR$(8) + CHR$(0) + CHR$(0) + CHR$(0)

PLAY WAVE

```
Syntax: PLAY WAVE WaveData$, volume
```

Play formatted sound data read from a string.

The supported WAVE parameters are:

- Uncompressed (PCM) or IMA 4-bit adaptive differential (IMA ADPCM). The ADPCM type is also known as DVI ADPCM; in a WAVE file, it's known as format 0x11.
- One or two-channels.
- All normal sampling rates (8k, 11k, 22.05k, 44.1k, 48, 96k).

You can't interrupt or abort a resource playback once it's been initiated. The resource always plays to the end of the data.

SCREEN LOCK

Syntax: SCREEN LOCK ON | OFF

Lock or unlock the current screen.

Palm OS 3.5 or greater is needed to use this command.

This command can be used to 'freeze' the display while doing lengthy drawing operations to avoid a flickering effect.
Call SCREEN LOCK OFF to unlock the display and cause it to be updated with any changes.
Because this command copies the screen, using it is a relatively expensive operation.

The screen will be temporally unlocked when calling commands like INPUT, OPEN FORM, INPUTBOX, etc. or when an error occurs.

List of Functions: Numeric Functions

ABS

```
Syntax: ABS( number )
```

Returns the absolute value of number, i.e. if number is negative, it's converted to positive.

ASC

```
Syntax: ASC( string )
```

Returns the ASCII code of the first character of string.
If string is empty, returns 0.

ATN

```
Syntax: ATN( number )
```

Returns arc-tangent of number.
Returned value is in radians. You can use DEG( ) to convert it to degrees.

MathLib library is required.

CALCDATETIME

```
Syntax: CALCDATETIME(year, month, day, hour, minute, second)
```

Return the number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the specified date and time.

COS

```
Syntax: COS( angle )
```

Returns cosine of angle, which must be in radians.
You can use RAD( ) to convert degrees to radians.

MathLib library is required.

CVD

```
Syntax: CVD( string )
```

Convert a 8-bytes string into a floating point number of double

precision. This function is the opposite of MKD$

CVI

Syntax: CVI(string)

Convert a 2-bytes string into an integer number. This function is the opposite of MKI$

CVL

Syntax: CVL(string)

Convert a 4-bytes string into an integer number. This function is the opposite of MKL$

CVS

Syntax: CVS(string)

Convert a 4-bytes string into a floating point number of single precision. This function is the opposite of MKS$

DAY

Syntax: DAY(seconds)

Returns the day (1 to 31) of a date specified as the number of seconds since January 1, 1904

DEG

Syntax: DEG(radians)

Convert radians to degrees.

DRIVE

Syntax: DRIVE(DriveName)

Returns -1 if a drive is available. Otherwise returns 0.
Use this function to check if a external SD card is inserted.

EOF

Syntax: EOF(file number)

Returns -1 (true) if the end of file has been reached.
If not, returns 0 (false).

ERR

Syntax: ERR

Return the number of error if the program is in an error-handling routine.

EXP

Syntax: EXP(number)

Returns number **e** raised to the power of number.

MathLib library is required.

FILEATTR

Syntax: FILEATTR(FileName$)

Returns the attributes of the specified file. If file doesn't exist, an error occurs. Use function DIR$ the check if the file exists.

The value returned by FILEATTR is the addition of the following attribute values:

- 0: Normal
- 1: Read only
- 2: Hidden
- 4: System file
- 16: Directory
- 32: Archive
- 256: Resources file (.PRC)

To know which attributes are set, use the relational operator AND to make a bit-to-bit comparison. For example:

```
    LET result = FILEATTR(FileName$)        AND 16
```

will return a value different to zero if FileName$ is a directory.

FIX

Syntax: FIX(number)

Returns the truncate part of number.
For example:

```
FIX(3.4) = 3
FIX(3.9) = 3
FIX(-3.4) = -3
FIX(-3.9) = -3
```

FRE

```
Syntax: FRE ( number )
```

Return the amount of free memory.

The meaning of returned value depends on value of number:

- -1: free dynamic memory.
- 1: total dynamic memory.
- -2: free space in stack.
- -3: free storage memory (RAM).
- 3: total storage memory (RAM).

HOUR

```
Syntax: HOUR(seconds)
```

Returns the hour (0 to 24) of a date specified as the number of seconds since January 1, 1904

IIF

```
Syntax: IIF( condition, TrueExpression, FalseExpression )
```

If condition is not zero, returns the true expression.
If it is zero, returns the false expression.

INSTR

```
Syntax: INSTR( start, string$, token$ )
```

Looks for token$ inside string$, starting at position number start.

If found, returns the position of token$ (1st character is 1).
If not found, returns zero.

INT

```
Syntax: INT( number )
```

Returns integer value of number, it is, the largest integer value

```
not greater than number.
For example:
INT(3.4) = 3
INT(3.9) = 3
INT(-3.4) = -4
INT(-3.9) = -4
```

LBOUND

```
Syntax: LBOUND( array, dimension )

Return the lower limit of an array.
If dimension is omitted, default dimension number is 1.

Returned value can be 1 or 0, depending on value of the field
Array base 1 in the menu option This program settings.
```

LEN

```
Syntax: LEN( string )

Returns the length (number of characters) of the string.
```

LOC

```
Syntax: LOC( file number )

Returns the current position for next reading or writing operation
of an open file.
```

LOF

```
Syntax: LOF( file number )

Returns the size of an open file.
If the file has been opened in RANDOM mode, or in RANDOM RESOURCE
mode, returns the number of records. Otherwise, returns the number
of bytes.
```

LOG

```
Syntax: LOG( number )

Returns natural logarithm (base-e) of number.

MathLib library is required.
```

MINUTE

Syntax: MINUTE(seconds)

Returns the minute (0 to 59) of a date specified as the number of seconds since January 1, 1904

MONTH

Syntax: MONTH(seconds)

Returns the month (1 to 12) of a date specified as the number of seconds since January 1, 1904

NOW

Syntax: NOW

Returns the number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the current date and time on the device.

NUMFILES

Syntax: NUMFILES(FileTemplate$ [CREATOR creator$ TYPE type$])

Returns the number of files or directories that match the specified pattern.

See function DIR$ for more details.

PI

Syntax: PI

Returns the value of pi: 3.1415…

POS

Syntax: POS(ScreenPos)

Return a screen value.

The meaning of returned value depends on value of ScreenPos:

- DRAWING X: current x-value for drawing operations (LINE TO…).
- DRAWING Y: current y-value for drawing operations.
- PRINT X: current x-value for PRINT operations.
- PRINT Y: current y-value for PRINT operations.

- FORM WIDTH: width of current form.
- FORM HEIGHT: height of current form.
- MAX WIDTH: max. width of screen.
- MAX HEIGHT: max. height of screen.

RAD

Syntax: RAD(degrees)

Convert degrees to radians.

RND

Syntax: RND

Returns a random float number between 0 and 1.
To change the seed, use command RANDOMIZE.

ROUND

Syntax: ROUND(number, decimals)

Returns a float number, rounded to the specified number of decimals.
If decimals is negative, the number is rounded to the left of decimal point.
For example:
ROUND(3.1415927, 3) = 3.142
ROUND(12345.678, -2) = 12300

RSRCID

Syntax: RSRCID(FileNumber, position)

Returns the resource ID of the record number position.
The file must be open as RANDOM_RESOURCE.

SECOND

Syntax: SECOND(seconds)

Returns the second (0 to 59) of a date specified as the number of seconds since January 1, 1904

SGN

Syntax: SGN(number)

Returns the sign of number.

Returned values are:
- 1 for positive values.
- 0 for zero values.
- -1 for negative values.

SIN

Syntax: SIN(angle)

Returns sine of angle, which must be in radians.
You can use RAD( ) to convert degrees to radians.

MathLib library is required.

SQR

Syntax: SQR(number)

Returns the square root of
number.

MathLib library is required.

STRCOMP

Syntax: STRCOMP(string1$, string2$, compare)

Returns -1, 0, or 1, based on the result of a string comparison.

The argument compare must be 0 for binary comparison, or 1 for text comparison.

Returned values are:
- -1 if string1$ < string2$.
- 0 if string1$ = string2$.
- 1 if string1$ > string2$.

SYSVAR

Syntax: SYSVAR (code)

Return the system numeric variable specified by code.

The available codes are:

- **OSVERSION** Returns a number which represents the OS version in hexadecimal format. For example, for OS version 3.10 the returned value is 784 (hexadecimal &H0310).
- **TICKS PER SECOND** Returns the number of ticks per second.
- **BATTERY** Returns the percentage of power remaining in the battery.
- **COLOR SCREEN** Returns -1 if the device has a colour screen, 0 otherwise.

TAN

```
Syntax: TAN( angle )

Returns tangent of angle, which must be in radians.
You can use RAD( ) to convert degrees to radians.

MathLib library is required.
```

TICKS

```
Syntax: TICKS

Returns the tick count since last reset.
The second count does not advance while the device is in sleep
mode.
```

TIMER

```
Syntax: TIMER

Returns the elapsed time (in seconds) since last reset.
The second count does not advance while the device is in sleep
mode.
```

UBOUND

```
Syntax: UBOUND( array, dimension )

Return the upper limit of an array.
If dimension is omitted, default value is 1.
For example, if the array a$ has been dimensioned as a$(10,3)
   UBOUND(a$, 1) is 10
   UBOUND(a$, 2) is 3
   UBOUND(a$) is 10
```

VAL

Syntax: VAL(string)

Convert a string to a numeric value.

This function allows only the dot '.' as decimal separator. To convert strings which are in local format, they must be delocalized first, with function DELOCALIZE$( ).

WEEKDAY

Syntax: WEEKDAY(seconds)

Returns the week day of a date specified as the number of seconds since January 1, 1904

The returned value means:

- 1: Sunday
- 2: Monday
- 3: Tuesday
- 4: Wednesday
- 5: Thursday
- 6: Friday
- 7: Saturday

YEAR

Syntax: YEAR(seconds)

Returns the year of a date specified as the number of seconds since January 1, 1904

List of Functions: String Funtions

CHR$

```
Syntax: CHR$( number )
```

Returns a one-character string, containing the character indicated in number.

CURDIR$

```
Syntax: CURDIR$
```

Returns the current directory.

DATE$

```
Syntax: DATE$
```

Returns a string containing the current date.

The format of that string can be changed selecting menu Options > This program settings, and changing the field Date format as local.

DELOCALIZE$

```
Syntax: DELOCALIZE$( string )
```

Delocalize a string containing a number.

Decimal separator and thousand separator are specified by system preferences.

DIR$

```
Syntax: DIR$(FileTemplate [CREATOR creator TYPE type])
```

Returns a string representing the name of a file or directory that matches the specified pattern.
If nothing matches, returns a zero-length string "".

The DIR$ function supports the use of multiple-character (*) wildcards to specify multiple files. However, single-character (?) wildcards are not supported.
You must supply a PathName the first time you call the DIR$ function. To retrieve the next item, you can make subsequent calls

to the NEXTFILE$ function.

ERROR$

Syntax: ERROR$(number)

Returns the error message that corresponds to a given error number.

FILECREATOR$

Syntax: FILECREATOR$(FileName)

Return a string representing the creator ID of the specified file.

See command OPEN for more details.

FILETYPE$

Syntax: FILETYPE$(FileName)

Return a string representing the type ID of the specified file.

See command OPEN for more details.

FORMAT$

Syntax: FORMAT$(number, format$)

Convert a numeric value to a string, with a specified format.

Available symbols in format$ string are:

- # Show one digit or nothing.
- 0 Show one digit or one zero.
- space Show one digit or one space.
- . Show the decimal point.
- , Show the thousands separator.
- E or e (at the end): Show the number in scientific notation.
- + or - (at the beginning): Show the sign at the beginning of string.

The order for symbols '#', '0' and space [ ] should be:

```
      ##[ ][ ]00 .00[ ][ ]##
```

it is, #, 0 and space in the integer part and space, 0 and # in the fractional part. If the order is wrong, the result string will be filled with #.

```
This function uses only the dot '.' as decimal separator. To
convert strings to local format, you can use function LOCALIZE$( )
afterwards.

Examples:
LET x# = 1234.567
FORMAT$(n, "+#,###.##")
   gives "+1,234.57".

FORMAT$(12.3, "[ ][ ][ ]0.00")
   gives "[ ][ ]12.30".
Note: [ ] represents a space.
```

HEX$

```
Syntax: HEX$( number )

Convert an integer number to a string representing its hexadecimal
value.
```

IIF$

```
Syntax: IIF$( condition, TrueExpression$, FalseExpression$ )

If condition is not zero, returns the true expression$.
If it is zero, returns the false expression$.
```

INKEY$

```
Syntax: INKEY$

Return a string containing last pressed key.
If there is no key, then return null string, i.e. ""
```

INPUT$

```
Syntax: INPUT$(length, FileNum)

Read data from a sequential open file, and returns these data as a
string. The file must have been opened in INPUT or BINARY mode.
```

LCASE$

```
Syntax: LCASE$( string )

Converts all characters in a string to lower case.
```

LEFT$

Syntax: LEFT$(string, quantity)

Returns a string containing a specified number of characters from the left side of the string.

LOCALIZE$

Syntax: LOCALIZE$(string)

Localize a string containing a number.

Decimal separator and thousand separator are specified by system preferences.

LTRIM$

Syntax: LTRIM$(string)

Returns a string containing a copy of a specified string without blank spaces at its left side.

MID$

Syntax: MID$(string, start, quantity)

Returns a string containing a number of characters specified by quantity, starting at character start (the first character of a string starts at 1, not at 0).

If quantity is not specified, then returned chararacters are from start until the end of the string.

MKD$

Syntax: MKD$(float)

Convert a float number into a 8-bytes string.
The number keeps the double precision.

MKI$

Syntax: MKI$(integer)

Convert an integer number into a 2-bytes string.
The number is first converted into 2-bytes integer.

MKL$

Syntax: MKL$(integer)

Convert an integer number into a 4-bytes string.

MKS$

Syntax: MKS$(float)

Convert a float number into a 4-bytes string.
The number is first converted into a single precision float.

NEXTFILE$

Syntax: NEXTFILE$

Returns a string representing the name of the next file or directory that matches the pattern that was previously specified with DIR$.
If no more files match, returns a zero-length string.

See function DIR$ for more details.

OCT$

Syntax: OCT$(number)

Convert an integer number to a string representing its octal value.

RIGHT$

Syntax: RIGHT$(string, quantity)

Returns a string containing a specified number of characters from the right side of the string.

RSRCTYPE$

Syntax: RSRCTYPE$(FileNum, position)

Returns the resource type of the record number position.
The file must be open as RANDOM_RESOURCE.

RTRIM$

Syntax: RTRIM$(string)

Returns a string containing a copy of a specified string without blank spaces at its right side.

SPC$

Syntax: SPC$(number)

Returns a string containing a specified number of blank spaces.

STR$

Syntax: STR$(number)

Convert a number to string.

This function uses only the dot '.' as decimal separator. To convert strings to local format, you can use function LOCALIZE$( ) afterwards.

STRING$

Syntax: STRING$(quantity, string)

Return a string with the character string repeated quantity times.

Only the 1st character of string is taken into account.

For example,
STRING$(5, "*") returns "*****"

STRING$(5, "abc") returns "aaaaa"

SYSVAR$

Syntax: SYSVAR$ (code)

Return the system string variable specified by code.

The available codes are:

- USERNAME Returns a string containing the name of the user which appears in HotSync operations.

TIME$

Syntax: TIME$

Returns a string containing the current time.

The format of that string is:
hh:mm:ss

TRIM$

Syntax: TRIM$(string)

Returns a string containing a copy of a specified string without blank spaces at its left and right sides.

UCASE$

Syntax: UCASE$(string)

Converts all characters in a string to capital letters.

List of GUI-Functions: Numeric Functions

ALERT

```
Syntax:
ALERT(alertID, text1, text2, text3)

Create a modal dialog from an alert resource and display the
dialog until the user taps a button in the alert dialog.

Up to three strings can be passed to this routine. They are used
to replace the variables ^1, ^2 and ^3 that are contained in the
message string of the alert resource.
```

CURFOCUS

```
Syntax: CURFOCUS

Return the ID of the object that has the focus

If no object has the focus, returns 0.
```

CURFONT

```
Syntax: CURFONT

Return the number of current font.

See command FONT for details.
```

CURFORM

```
Syntax: CURFORM

Returns the ID number of the currently active form
```

CURINK

```
Syntax: CURINK

Return the current foreground colour.

See command INK for details.
```

CURPAPER

```
Syntax: CURPAPER
```

Return the current background colour.

See command PAPER for details.

CURRESOLUTION

Syntax: CURRESOLUTION

Return a value that represents the current resolution of the screen.

Returned values for density are:

- 72 = STANDARD
- 108 = ONE AND A HALF
- 144 = DOUBLE
- 216 = TRIPLE
- 288 = QUADRUPLE

See command RESOLUTION for details.

FONTHEIGHT

Syntax: FONTHEIGHT

Returns the height in pixels of a line in the current font.

The height of a line is the height of the character cell plus the space between lines (the external leading)

FONTWIDTH

Syntax: FONTWIDTH(string)

Returns the width of the string, in pixels.

The missing character symbol (an open rectangle) is substituted for any character that does not exist in the current font.

GROUPSELECTION

Syntax: GROUPSELECTION(group)

Returns the object ID of the selected control.
If no item is selected, returns 0.

LASTEVENT

Syntax: LASTEVENT(value)

Return the data for the event that occured with some commands like WAIT, INPUT, WAIT BTN, INKEY…

The available data to return are:

- EVENT TYPE: type of event. The most common types are:
  - 0 (nilEvent)
  - 4 (keyDownEvent)
  - 9 (ctlSelectEvent)
  - 10 (ctlRepeatEvent)
  - 12 (lstSelectEvent)
  - 14 (popSelectEvent)
  - 21 (menuEvent)
  - 22 (appStopEvent)
  - 33 (sclExitEvent)
- LAST KEY: if the event-type is a keyDownEvent, returns the code of the pressed key
- OBJECT ID: if the event is related to an object, returns the object ID
- PEN DOWN: returns -1 (true) if the pen was down at the time of the event, otherwise 0 (false)
- SCREEN X or SCREEN Y: window-relative position of the pen in pixels (number of pixels from the top-left bound of the window)
- TAP COUNT: the number of taps received at this location. This value is used mainly by fields. When the user taps in a text field, two taps selects a word, and three taps selects the entire line
- DATUM0 to DATUM7: the specific data for an event, if any. Its contents depend on the event type

See PalmOS documentation for details.

LISTSELECTION

Syntax: LISTSELECTION(objectID)

Returns the currently selected choice in the list.

The list choices are numbered sequentially.
The first item can be 1 or 0, depending on value of the field Array base 1 in the menu option This program settings, it is, the value returned by LBOUND().
If none of the items are selected, returns -1 for ArrayBase 0 or 0

for ArrayBase 1.

MSGBOX

Syntax: MSGBOX(prompt, [buttons, title])

Displays a message in a dialog box, waits for the user to click a button, and then returns an integer indicating which button the user clicked.

- prompt is a string expression displayed as the message in the dialog box.
- buttons is an optional numeric expression specifying the number and type of buttons to display.
  - OK ONLY: Displays only the OK button.
  - OK CANCEL: Displays OK and Cancel buttons.
  - ABORT RETRY IGNORE: Displays Abort, Retry and Ignore buttons.
  - YES NO CANCEL: Displays Yes, No and Cancel buttons.
  - YES NO: Displays Yes and No buttons.
  - RETRY CANCEL: Displays Retry and Cancel buttons.

  If omitted, the default value is OK ONLY.
- title is an optional string displayed in the title bar of the dialogue box.

NUMFORMS

Syntax: NUMFORMS

Return the number of open forms.

NUMOBJECTS

Syntax: NUMOBJECTS

Return the number of objects in the current form.

OBJDYNAMIC

Syntax: OBJDYNAMIC(objectID)

Returns -1 (true) if the object has been created dynamically. Otherwise, returns 0 (false).

OBJID

Syntax: OBJID(ObjectIndex)

Returns the object ID, given its index number.

The index of the first object is 1, and the index of last object is the same than the value returned by the function NUMOBJECTS. If no object corresponds to given index, returns -1.

OBJNDX

Syntax: OBJNDX(objectID)

Return the index number of the specified object in the in the current form's objects list.

The index of the first object is 1, and the index of last object is the same than the value returned by the function NUMOBJECTS.

OBJPOS

Syntax: OBJPOS(objectID, value)

Return a value related to the position of the specified object.

The available choices for value are:

- OBJ X: returns the window-relative X coordinate
- OBJ Y: returns the window-relative Y coordinate
- OBJ WIDTH: returns the width of the object
- OBJ HEIGHT: returns the height of the object

OBJSTYLE

Syntax: OBJSTYLE(objectID)

Return the style of the specified object.
This function only works for objects created dynamically.
Otherwise, returns -1.

See the PalmOS documentation for details.

OBJTYPE

Syntax: OBJTYPE(objectID)

Return the type of the specified object.

The meaning of the returned value is:

- 0 Field

- 1 Control
- 2 List
- 3 Table
- 4 Bitmap
- 8 Label
- 9 Form title
- 10 Popup list
- 11 Graffiti shift indicator
- 12 Gadget (custom object)
- 13 Scrollbar

See the PalmOS documentation for details.

OBJVALUE

Syntax: OBJVALUE(objectID)

Return the value of the specified object.

For scroll bars and sliders, return its value.
For push buttons and check boxes, return -1 (true) if control is selected, or zero (false) if unselected.

POPUPLIST

Syntax: POPUPLIST(objectID)

Display a modal window that contains the items in the list and returns the list item selected, or 0 if no item was selected.

RGB

Syntax: RGB (red, green, blue)

Returns an integer value representing an RGB colour value from a set of red, green and blue colour components.

WAIT BTN

Syntax: WAIT BTN

Waits until a control is pressed, and return its object ID.

List of GUI-Functions: String Functions

INPUTBOX$

```
Syntax:
INPUTBOX$(prompt, buttons, [title, default])

Displays a prompt in a dialog box, waits for the user to input
text or click a button, and then returns a string containing the
contents of the text box.

prompt is a string expression displayed as the message in the
dialog box.
title is an optional string displayed in the title bar of the
dialog box.
default is an optional string displayed in the text box as the
default response if no other input is provided. If you omit
default, the displayed text box is empty.
```

OBJTEXT$

```
Syntax: OBJTEXT$(objectID)

Returns the value of of the specified object.

For fields, returns its current value.
For controls, returns its current label.
For a list, returns the text of the selected item. If no item is
selected, returns an empty string ''''.

For other kind of object, generates an Invalid Object error.
```

List of Errors

Error 1: Syntax error

There is a syntax error in written line.
To check where, open the line with error and then try to save it.
It will appear the symbol for syntax error: '??'

Error 2: Break

A STOP command has been found, or the application launcher silk-button has been pressed in order to stop the program.

Error 3: Internal error

An internal fatal error has occurred. Execution will be stopped.

Error 4: Not enough free memory

There is not enough free memory to work.

Error 5: Stack overflow

Stack has been overflowed.
One reason can be that the stack size is too small, but usually it is due to a user function which is repeatedly called.
You can check the Calls Stack.

Error 6: String too long

The size of the string is too large, or there is not enough free memory.
The maximum size of a string is 63.5 KB (65024 bytes).

Error 7: Out of string space

There is no more space to create a new string.

Error 8: Invalid argument

A function has been called with an invalid argument.

Error 9: Invalid array index

The passed index is out of range, or the number of indexes is not correct.

Error 10: MathLib not present

A mathematical function has been called, and the MathLib library is not present.
MathLib is a library that some other programs can use.
It is freeware and it can be downloaded from internet at http://www.radiks.net/~rhuebner/mathlib.html

Error 11: Not a number

An arithmetic operation caused the NaN error. The most common reasons are:

- SQR of a negative number
- a division 0 / 0
- a division INF / INF
- a multiplication INF * 0

Error 12: Number too big

The result of an arithmetic operation is out of limits.
The limit for an floating point operation is: ±1.79769E+308
The limits for an integer operation are:

- +2,147,483,647 for positive values.
- -2,147,483,648 for negative values.

Error 13: Division by zero

Program tried to divide a number by zero.

Error 14: Invalid name

A file command or function contains an invalid name.

Error 15: Invalid file number

A file command or function contains an invalid file number. Maybe the file number is out of limits (between 1 and 255) or the file hasn't been open.

Error 16: File not found

The file doesn't exist.
Use the function DIR$() to check whether a file exists or not.

Error 17: File not valid

The type of the file is not correct.
Try to use the separator CREATOR and TYPE if they are not the default.

Error 18: File number already in use

Command OPEN used with a file number which is already in use.

Error 19: File already exist

Commands FILECOPY, NAME or MKDIR with a destination file name that already exists.
Use the function DIR$() to check whether a file exists or not.

Error 20: File already open

Command OPEN tried to open a file which is already open.

If the file remained open due to a program crash, try to re-install it.

Error 21: Could not create file

Commands OPEN or FILECOPY failed when trying to create a new file.

Error 22: File mode not valid

Command OPEN tried to open an existing file in a mode that the file doesn't accept.
Try to use the separator CREATOR and TYPE if they are not the default.

Error 23: Invalid record number

Commands PUT or GET tried to work with a record number out of limits.
Check the number of existing records with function LOF().

Error 24: Storage area is full

There is no more space in the storage are.
Check the amount of free space with function FRE(-2).

Error 25: Permission denied

Write permission is not enabled for that file.
Use function FILEATTR() to check permissions of a file.

Error 26: Input past end

Program tried to read data from a file (with INPUT #, GET # …), and there are no more data to read.
Use function EOF() to check if the end of a file has been reached.

Error 27: Compressed documents are not supported

Program tried to read data from a sequential file (with INPUT #, LINE INPUT # …), and that file has been saved as a compressed document.

Error 28: Device error

This error is reserved.

Error 29:

This error is reserved.

Error 30: File error

It occurred an undetermined error with a file.

Error 31: Device I/O error

It occurred an Input/Output error.

Error 32: Device unavailable

The program tried to use a device which is unavailable.
Use function DRIVE() to check if a device is available.

Error 33: Invalid Rom Version

The program called a command or a function that needs a higher Palm OS version.
The OS version can be checked with function SYSVAR()

Error 34: RETURN without GOSUB

A command RETURN has been found when no GOSUB has been called before.

Error 35: RESUME without ERROR

A command RESUME has been found when no error has been occurred or ON ERROR hasn't been programmed before.

Error 37: Type mismatch

A numeric value was found when it was expected a string, or vice versa.
The most common causes are:

- CASE, whit different type of SELECT CASE
- TEXT OBJECT, trying to modify a numeric field with a non-numeric string

Error 38: Invalid object number

A GUI-command or a GUI-function was called with the object ID out of limits.
The object ID must be an integer value between 0 and 8999.

Error 39: Object number already exists

A new object was tried to be created, and there already is another one with the same ID.

Error 40: Object not found

The object ID doesn't exist is current form.

Error 41: Array is used as an options-list in a GUI object

You tried to REDIM a string-array which is used as an options-list in a list or in a trigger.

Error 42: Invalid object type

The object doesn't allow the operation that you tried to do. Use the function OBJTYPE() to know the object type.

Error 43: Form not found

You tried to open a form that doesn't exist in the resources file.

Error 44: Form already open

You tried to open a form that is already open.

Error 45: Screen already locked

You tried to lock the screen, and it's already locked

Error 46: Not available in Demo mode !

The command or the function that was called is not available in Demo mode.

Error 47: Command not available in this version

The command that was called is not available in this version.

Error 48: Function not available in this version

The function that was called is not available in this version.

Error 257: IF without ENDIF

It was found an IF command without its corresponding ENDIF. Press the trigger at the left of the line and select 'Search ending line'

Error 258: ELSE without ENDIF

It was found an ELSE command without its corresponding ENDIF. Press the trigger at the left of the line and select 'Search ending line'

Error 259: ELSE without IF

It was found an ELSE command without its corresponding IF. Press the trigger at the left of the line and select 'Search beginning line'

Error 260: END IF without IF

It was found an ENDIF command without its corresponding IF. Press the trigger at the left of the line and select 'Search beginning line'

Error 261: WHILE without WEND

It was found a WHILE command without its corresponding WEND.

Press the trigger at the left of the line and select 'Search ending line'

Error 262: WEND without WHILE

It was found a WEND command without its corresponding WHILE.
Press the trigger at the left of the line and select 'Search beginning line'

Error 263: DO without LOOP

It was found a DO command without its corresponding LOOP.
Press the trigger at the left of the line and select 'Search ending line'

Error 264: LOOP without DO

It was found a LOOP command without its corresponding DO.
Press the trigger at the left of the line and select 'Search beginning line'

Error 265: EXIT DO is not inside a DO loop

Command EXIT DO was programmed outside its corresponding DO..LOOP loop.

Error 266: FOR without NEXT

It was found a FOR command without its corresponding NEXT.
Press the trigger at the left of the line and select 'Search ending line'

Error 267: NEXT without FOR

It was found a NEXT command without its corresponding FOR.
Press the trigger at the left of the line and select 'Search beginning line'

Error 268: EXIT FOR is not inside a FOR-NEXT loop

Command EXIT FOR was programmed outside its corresponding FOR…NEXT loop.

Error 269: FOR-NEXT variables don't match

The variable programmed with NEXT is not the same than the one programmed with FOR.
Press the trigger at the left of the line and select 'Search beginning line' to locate the line that contains the command FOR.

Error 270: SELECT CASE must be followed by a CASE clause

The next line following a SELECT CASE sentence is not a CASE clause.

Error 271: SELECT CASE without END SELECT

It was found a SELECT CASE command without its corresponding END SELECT.
Press the trigger at the left of the line and select 'Search ending line'

Error 272: END SELECT without SELECT CASE

It was found a END SELECT command without its corresponding SELECT CASE.
Press the trigger at the left of the line and select 'Search beginning line'

Error 273: CASE without SELECT CASE

It was found a CASE command outside its corresponding SELECT CASE .. END SELECT.

Error 274: GOTO beyond loop limits

The line where to go is outside the current loop.
Commands EXIT DO or EXIT FOR are recommended for ending a loop.
If you need to program this GOTO, you can avoid this error selecting the checkbox 'Ignore GOTO errors' in Menu>This program settings.

Error 275: Constant not defined

The named-constant you try to use, hasn't been defined with command CONST.

Error 276: Constant not defined

The named-constant you try to use, hasn't been defined with command CONST.

Error 277: Constant already defined

The named-constant you try to define, has already been defined with command CONST.

Error 278: Constant already defined

The named-constant you try to define, has already been defined with command CONST.

Error 279: Not all constants have been defined

There is a named-constant that hasn't been defined with command CONST.
The program will continue without any problem, but undefined numeric constants will be set to zero and string constants to zero-length string "".
If a constant is not used, it can be deleted pressing the button

VAR at the bottom of the screen, then selecting it and pressing the delete button.
Another way to delete all unused constants and variables is Menu>Run>Clean variables space.

Error 280: Not all constants have been defined

There is a named-constant that hasn't been defined with command CONST.
The program will continue without any problem, but undefined numeric constants will be set to zeroand string constants to zero-length string "".
If a constant is not used, it can be deleted pressing the button VAR at the bottom of the screen, then selecting it and pressing the delete button.
Another way to delete all unusued constants and variables is Menu>Run>Clean variables space.

Error 281: Command CONST must be at the beginning of the program

The command CONST is allowed only at the beginning of the program, before any executable line.

Error 282: Command CONST is not allowed inside SUBs

The command CONST is allowed only at the beginning of the program, before any executable line, and only in the MAIN PROCEDURE.

Error 283: EXIT SUB is not inside a SUB

The command EXIT SUB has been programmed in the MAIN PROCEDURE.

Error 284: Invalid number of indexes

The number of indexes that you have passed to the array, is not the same than the defined one.
You can check the number of indexes pressing the button VAR and then selecting the array.

Error 287: Invalid number of paramaters

The number of parameters that you passed to a function or to a command doesn't match the defined one.

Error 290: Initial Form not found !

The initial form that is defined in Menu>This program settings>Next doesn't exist is the resources file.

Error 321: Can't continue (a previous Fatal Error had occurred)

Some errors prevent continuing, because the result can be unpredictable.
These errors are:

- 3 Internal error
- 4 Not enough free memory
- 5 Stack overflow