iziBasic v6.0

November 4, 2005

Index

iziBasic syntax

Legend:

n is Number or Defined Constant
v is NumVar (A-Z or a user defined NumVar with the DIM statement)
f is NumFunction
c is TextVar (A$-Z$ or a user defined TextVar with the DIM statement)
t is Text
s is TextFunction

v|n is either NumVar, Number or Defined Constant
c|t|s is either TextVar, Text or TextFunction
and so on...

Notes:

- v|n are 32 bit float IEEE 754 compatible numbers
- numbers can be passed in integer format ([-]n[n][..]), in float format ([-]n[n][…].n[n][..]) or in exponential format ([-]n.nnnnnnne[-]nn)
- c|t are strings of characters with up to 63 characters
- character “¶” is CR/LF within c and t

Variables assignment and calculation

In statements, v|n or c|t cannot be an aggregate of calculations.
So, in iziBasic, you should work this way (with an example):

```
C=3*COS(B)+5 : D=MAX(A,B)
IF C < D PRINT “OK”
E=MIN(A,B) : E=5*MAX(C,E)
```

When with some other Basic compilers or interpreters you could do:

```
IF 3*COS(B)+5 < MAX(A,B) PRINT “OK”
E=5*MAX(C,MIN(A,B))
```

Following is a list of the compiling directives, statements and functions available in iziBasic.
As iziBasic is a subset of the BASIC language, with just a few specificities, I have only included a quick explanation of what the statements and functions do. Please refer to a BASIC documentation if you are not familiar with the BASIC language.

ⓘ Compiling directives, statements and functions in **GREEN** (also marked with a little $ glyph for the unlucky ones who would like to print this manual and do not have a color printer) are only available in the full version of iziBasic.

Compiling directives

{CONSOLEFONT ON|OFF}

Forces console fonts (low resolution and high resolution) to be included in an application or to be removed.

Notes:
- The sizes of console fonts are 2598 bytes for the low resolution font and 4280 bytes for the high resolution font. So, setting this directive to OFF, your application may have a smaller footprint by about 7 Kbytes.
- See the SETFONT statement

{CREATORID t}

Sets application CreatorID (4 characters).

Notes:
- This directive is mandatory.
- As all Palm OS software, your application should have a unique 4 characters Creator ID. Please refer to the Palm OS website to get all information about this Creator ID "*thing*" and to register yours.

$ {DEFINE t}, **{IFDEF t}**, **{IFNDEF t}** and **{ENDIF}** conditional compiling directives

Allows you define specific labels ({DEFINE t}) and conditional compiling of source code blocks (from a {IFDEF t} or a {IFNDEF t} to a {ENDIF}) according to if the t label is defined ({IFDEF t}) or not ({IFNDEF t}).

Notes:
- Conditional compiling directives cannot be overlapping
- These directives are managed like full statements by the iziBasic compiler, so they cannot be put within a statement. Example:
  OK {IFDEF "HaHa"} A=1 {ENDIF} {IFNDEF "HaHa"} A=2 {ENDIF}
  Not OK A = {IFDEF "HaHa"} 1 {ENDIF} {IFNDEF "HaHa"} 2 {ENDIF}
- Only the 19 first characters of a t label are taken into account by iziBasic

$ {INCLUDE t }

Include one source code into another one and have the iziBasic compiler parse the source codes accordingly. This is very convenient for adding a set of routines that you want to use in different development projects.

Note:
- One included source code file cannot itself include other source codes files. In other words, includes cannot be nested. So, only your top source code may include other files.

$ {KEYEVENTS ON|OFF|PARTIAL}

Defines whether you want to manage all hard buttons (of the device) events, none of them or some of them in the DOEVENTS and WAITEVENT functions. By default, this directive is set to OFF.

Note:
- Be careful that, when this directive is set, iziBasic overrides the normal behavior of these buttons presses, meaning that if you, for instance, press the Date Book hard button, exiting from iziBasic will not automatically launch the Date Book application.
- The PARTIAL parameter allows tracking all hard buttons except for the 4 buttons assigned to launching applications (by default: Address Book, Memo Pad…)
- In the case of the use of the CHAIN statement, this directive should be set in all source code blocks.
- See the explanations for the DOEVENTS function

{MINOSVERSION t}

Set the minimum Palm OS version (format “M.m”, where M is Major and m minor) for your application to run. It cannot be smaller than “3.0”.
If it is set, at runtime your application will check if the target device meets this minimum OS requirement and quit smoothly with a message if it is not met.

Notes:
- This directive is facultative. If not set, iziBasic will assume that it is worth “3.0”.
- The iziBasic runtime checks the device’s Palm OS version and these instructions are executed only if the test is positive, otherwise they are ignored.
- You should test that your software made with iziBasic will work on all targeted devices, just like with all other development tools.

(…continued on next page…)

ⓘ Warning: I cannot guarantee that all YOUR iziBasic developments will run smoothly from Palm OS 3.0 on. Indeed, iziBasic uses many Palm OS API calls (which are embedded for your convenience) that were introduced progressively over the Palm OS versions. I have identified in this manual the few instructions which require a minimum Palm OS version with informative panels:

ⓘ As an example of specific incompatibility that iziBasic cannot cope with for you, please refer to the DESTROY statement in the GUI module (read Handspring note).

MINOSVERSION also defines which size for Code, Numbers and Text stacks is made available.

This size is limited by the devices' available dynamic RAM, itself being highly fluctuating among devices and it also changes according to the total RAM storage within a device type. So, easing your work was not easy in this area but I nevertheless built the following grid (which I hope will work in all cases!) even though it does not optimize all possibilities.

Palm OS version	< 3.5	3.5 to 4.n	3.5 to 4.n	5.n
Device's RAM	>= 2 MB	< 4 MB	>= 4 MB	>= 4 MB
Code Stack	4000	4000	6000	24000
Numbers Stack	255	255	1000	12000
Text Stack	200	200	400	800

iziBasic itself and programs made with it do not check the device's total RAM. They are more precise than that: they check for the available dynamic RAM (as some other software running in the background might already have reserved part of it) and will exit smoothly with an information message if there is not enough of it.

ⓘ You have to run iziBasic itself on a device which has a sufficient memory capacity to fulfill these requirements (in other words that has enough RAM). For instance: if you run iziBasic on a device with Palm 0S 3.3 and set MINOSVERSION to "5.0", the stacks sizes available in your application will be those for Palm OS <3.5

Notes:

- Most devices currently in use are to be found in the 2 columns on the right side of the previous grid and there should be no trouble with them.
- There is a tricky case for devices with Palm OS 3.5 to 4.n and with less than 4 MB of RAM. Would you wish your application to run on these devices and not on earlier devices, set MINOSVERSION to a fake "3.4" value (as Palm OS 3.4 was never made public!). If you set it with a "3.5" or bigger value, the case of >= 4 MB of RAM will be considered by iziBasic.
- See appendix #8.

{PARSER ON|OFF}

Decide how you want to manage Mathematical parsing capabilities according to operations priorities in expressions. By default, this directive is set to OFF.

Notes:

- If PARSER compiling directive is set to ON: 1+2*3 = 7, this is is 1+(2*3)
  or in other words: the operations are made according to the operations priority order:
  Priority #1: parenthesis
  Priority #2: functions
  Priority #3: ^ (exponentiation)
  Priority #4: * / \ MOD
  Priority #5: + -
  Priority #6: <= = <> = > >=
  Priority #7: AND OR XOR
  Within a given priority, operations are made from left to right.
- If PARSER is ON, you may also use parenthesis to force the order of operations in the variables assignment: (1+2)*3 = 9
- If PARSER compiling directive is set to OFF: 1+2*3 = 9, this is (1+2)*3 (this was the operational mode in previous releases of iziBasic)
  or in other words: the operations are made from left to right
  Parenthesis cannot be used in this mode.
- PARSER compiling directive can be set and unset all along the source code, the last status is remembered for further variables assignments:
  {PARSER ON}
  A=1+2*3 ' A=7
  {PARSER OFF}
  B=1+2*3 ' B=9
- In terms of performance, iziBasic's compiled code is smaller and faster when PARSER is set to OFF
- In the compilation report, the PARSER memory usage in the Numbers stack is reported in between brackets. Example: `Number Size: 125[+21] / 4000`

$ {RESOURCEFILE [+] t}

Add a resource file to your program. This is, for instance, very useful to add images (of Tbmp type) which will then be used by the IMAGE and IMAGEBUTTON instructions.

Notes:
- This directive is facultative.
- In your PRC file, the default images shipped with iziBasic are of Tbmp type and are numbered from 1 to 40. Please refer to the IMAGEBUTTON instruction to see the list of these 40 available images.
- The [+] facultative parameter lets you decide if you want to include your resources in addition to the 40 images shipped with iziBasic. If not set, the 40 default images will not be included to your program.
- You may build your resource files with a third party software (see Appendix #4).

$ {SECUREFILES ON|OFF}

To avoid any risk of deleting important files on your Palm device, the OUTPUT, APPEND and RANDOM file modes of the files OPEN statement only work with databases having a “LDIB” Creator ID if this compiling directive is set to ON.
If set to OFF, it can write to any database at runtime. If the database is to be created by OPEN, it will then be with the Creator ID of the application as defined by the CREATORID compiling directive.
By default, this directive is set to ON.

Note:
- In the case of the use of the CHAIN statement, this directive should be set in all source code blocks.
- See the OPEN statement in the Files module.

{VERSION t}

Sets application version.

Note:
- This directive is facultative.

Math Operators and precedence

iziBasic recognizes the following operators, with their level of precedence given (1 = lowest and 6 = highest):

-	6	unary minus sign
^	5	exponentiation
*	4	multiplication
/	4	division
\	4	integer division
MOD	4	modulus (remainder) arithmetic
+	3	addition
-	3	subtraction
=	2	equality
<>	2	inequality
<	2	less than
>	2	greater than
<=	2	less than or equal to
>=	2	greater than or equal to
AND	1	conjunction
OR	1	disjunction
XOR	1	exclusive or

Test Operators

=	equality
<>	inequality
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Labels label:

Notes:

- As from version 5.0 of iziBasic, labels are no more case sensitive, therefore LABEL is now the same thing as Label or label
- You can put up to 255 labels and user defined variables in one source code
- Only the 20 first characters of a label are taken into account by iziBasic

Advice:

- Make sure that none of your labels starts with a reserved keyword. For example: CloseMyForm: or GoToMySubRoutine: are not good (and will lead to strange behaviors either at compilation time or at execution time!) when MyFormClose and JumpToMySubRoutine are valid. A good practice / trick to make sure this does not happen is to prefix all your labels with a set of characters like “lbl” (which stands for LaBeL) or the “_” (underscore) character: lblCloseMyForm or _GotoMySubRoutine are valid labels.

Statements SingleStatement [: SingleStatement] [...]

Notes:

- a SingleStatement MUST be < 62 characters long
- statements are not case sensitive

Core Statements

These are the “core” statements, very similar to the other BASIC dialects.

BEEP [v|n_1] [, v|n_2]

Generates a beep sound.

Notes:
- the v|n_1 parameter is the number of beeps to play. If not set, beep once
- - the v|n_2 parameter is the type of sound to play (if not set, the Info beep is played):

  1 = Info 2 = Warning 3 = Error 4 = StartUp
  5 = Alarm 6 = Confirmation 7 = Click

BEGIN
Statement
END

Entry point and last instruction of the program.

Notes:
- both BEGIN and END must be defined
- only the last BEGIN is taken into account by iziBasic if you input several of them. But you can have several END instructions.

BREAK

Break program execution (for debugging purposes...)

CALL v|n

Call an assembler-like subroutine at address v|n in the code stack.

Notes:
- not to be used until I provide a documentation of how to build assembler-like routines in the Code Stack
- meanwhile, you can use PEEK and POKE to store [0..65535] values in the Code Stack, making sure you do not override the used Code Stack (see FRE function)

$ CHAIN t

The program execution will chain to the “NameOfProgram.ibas” given in t, starting at the BEGIN statement of “NameOfProgram”.

Notes:

- the Code Stack will be emptied and refilled with the new program code which will start at its BEGIN point
- Numbers and Texts Stacks are not emptied, so values are passed to the new code. Would you wish to empty them too, just use the CLEAR statement.
- But the MegaString is emptied, so, if you need it in the new code segment, you should save its content to a temporary file before chaining and reload it from this file in the new code segment.

CLEAR [v-v | c-c]

Clear all A-Z NumVars (set to 0) and A$-Z$ CharVars (set to empty text string “”), a range of Numvars only, or a range of CharVars only

CONST c = t

Define c as a text constant.

Note:

- Doing C$=”Some text” will reserve 1 space in the Text stack, when CONST C$=”Some text” will only require no extra space but C$ which is always reserved.

CONST v = n

Define v as a numerical constant.

Note:

- Doing C=123 will reserve 1 space in the Number stack, when CONST C=123 will only require no extra space but C which is always reserved.

DEC v

Decrements v by one unit (equivalent to v = v - 1).

$ DIM %var%|%var$ [, %var%|%var$] […]

Defines one or several custom Number or Text variable(s), in addition to A-Z and A$-Z$ which are automatically defined.

Notes:

- A user defined variable must be prefixed with the % character and end with the % character for Number variables or the $ character for Text variables
- A user defined variable should not have more than 20 characters, including the prefix character (%) and the ending character (% or $)
- User defined variables are not case sensitive, therefore %MyVar$ is the same thing as %MYVAR$ or %myvar$
- You can put up to 255 user defined variables and labels in one source code
- DIM must be defined before the iziBasic compiler reserves some space in the Numbers stack for its use, so you should place your DIM at the top of your program, just after the Compiling Directives
- See Appendix #8

DO
Statement
LOOP v|n TestOper v|n or **LOOP c|t TestOper c|t**

DO implements a number of loops. The program will exit from the loop when the condition after the LOOP statement is met.

Notes:

- equivalent to REPEAT / UNTIL
- the statement in loop will be executed at least once (on the difference of WHILE / WEND which check the condition before looping)

GOSUB label

GOSUB initiates a subroutine call to the specified label. The subroutine must end with RETURN. It will then carry on with the statement following the GOSUB one.

GOTO label

GOTO branches program execution to the specified label

label:
Statement
RETURN

RETURN concludes a subroutine called by GOSUB to the specified label.

FOR v = v|n_1 [DOWN]TO v|n_2 [STEP v|n_3]
Statement
NEXT

FOR initiates a FOR / NEXT loop with the variable v initially set to v|n_1 and incrementing (TO) or decrementing (DOWNTO) in v|n_3 steps (default is 1 for TO and -1 for DOWNTO) until v equals v|n_2.

IF v|n TestOper v|n Statement or **IF c|t TestOper c|t Statement**

IF evaluates the given expression and performs the Statement if it is true.

Notes:
- the Statement can be any single statement or multiple statements (separated by the ":" character) but a non completed "block statement" (like IF THEN [ELSE] END IF, SELECT CASE / END SELECT statement, REPEAT / UNTIL, …). Indeed this syntax of the IF statement is equivalent to this one: IF test THEN Statement : ENDIF (see below), so there is a risk of bad indentation of the quiet ENDIF of this syntax.
- Multiple IF statements can be imbricated this way: IF test1 IF test2 Statement. This is similar to: IF test1 AND test2 Statement (which is not valid in iziBasic's syntax)

IF v|n TestOper v|n THEN Statement or **IF c|t TestOper c|t THEN Statement**
[ELSE Statement]
END IF

IF evaluates the given and performs the THEN statement if it is true or (optionally) the ELSE statement if it is FALSE, ending with END IF.

Note:
- Statements can be put on the next line after THEN and ELSE. Example:

```
IF A=1 THEN B=2 : C=3
ELSE B=3
END IF
```

```
IF A=1 THEN
  B=2 : C=3
ELSE
  B=3
END IF
```

INC v

Increments v by one unit (equivalent to v = v + 1).

[LET] c = c|t|s [+ c|t|s] [...]

LET assigns the value of the expression after the "=" sign to the variable c.
iziBasic supports implied LET statements, meaning that the LET statement is facultative.

Note:
- s can also be a A$(v|n), please read the Arrays paragraph

[LET] v = v|n|f [MathOper v|n|f] [...]

LET assigns the value of the expression after the "=" sign to the variable v.
iziBasic supports implied LET statements, meaning that the LET statement is facultative.

Notes:
- please see the PARSER compiling directive
- f can also be a A(v|n), please read the Arrays paragraph

POKE $v|n_1$, $v|n_2$

Puts value $v|n_2$ at address $v|n_1$ in the code stack

Notes:
- $v|n_1$ should be in the [FRE(0)..FRE(3)] range. Be careful not to POKE under the address returned by the FRE(0).function as it will corrupt the compiled code by iziBasic and your program will behave in a strange manner if you do so!
- $v|n_2$: has to be a [0...65535] value.

POP c|v

Returns top Text stack value in the c variable, or the top Number stack value in the v variable. The Text or Number stack is then decremented by one unit, meaning that it then points to the last but one top value.

PUSH c|t|v|n

Increments the Text or Number stack by one unit.
It then puts the c|t value to the Text stack, or the v|n value to the Number stack.

REM or '

REM and ‘ allow remarks to be included in a program. As currently implemented, the entire remaining text on a line following REM (or ‘) is ignored by the iziBasic compiler.

Notes:

- A REM comment should be preceded by a statement delimiter (“:”) if not at the beginning of a line; this is not required for the ‘ format.
- It is a good habit to comment as much as possible your source code, it allows easier reading back and understanding of it later on!

REPEAT
Statement
UNTIL v|n TestOper v|n or **UNTIL c|t TestOper c|t**

Equivalent to DO / LOOP, please refer to the DO / LOOP statement.

SELECT CASE v|c
CASE n_1|t_1
Statement
[…]
[CASE n_n|t_n
Statement]
[CASE ELSE
Statement]
END SELECT

SELECT CASE introduces a multi-line conditional selection statement. The v (or c) variable given as the argument to SELECT CASE will be evaluated against numerical (or text) values by the CASE statements following. If no evaluation was successful, the CASE ELSE statement will be executed. The SELECT CASE statement concludes with an END SELECT statement

Notes:

- only the first true condition will be executed so make sure the conditions are unique
- you may have up to 255 CASE items in a SELECT CASE / END SELECT
- up to 89 SELECT CASE / END SELECT statements can, as from version 6.0 of iziBasic, be imbricated one into another, meaning that up to 89 successive CASE items may include themselves a full SELECT CASE / END SELECT group

SLEEP v|n

Does nothing during v|n seconds

Note:

- would you wish to have your program wait for less than one second, then use a routine with the TICKS and TICKSPERSEC functions

SWAP v , v

As its name states it, SWAP swaps the values of two variables.

SWAP c , c

As its name states it, SWAP swaps the values of two variables.

WHILE v|n TestOper v|n or **WHILE c|t TestOper c|t**
Statement
WEND

WHILE implements a number of loops, delimitated with the WEND statement. The program will exit from the loop as soon as the condition after the WHILE statement is met.

Note:

- on the difference of DO / LOOP and REPEAT / UNTIL the condition is checked before looping. So it might never be executed if the condition is met at the very beginning.

Core Functions

These are the “core” functions, very similar to the other BASIC dialects.

NumFunctions:

ABS(v|n)

Returns the absolute value of the argument v|n

ACOS(v|n)

Returns the arc cosine value of the argument v|n

ASC(c|t)

Returns the ASCII code for the first letter in the argument c|t

ASIN(v|n)

Returns the arc sine value of the argument v|n in radians

ATAN(v|n)

Returns the arc tangent value of the argument v|n in radians

COS(v|n)

Returns the cosine value of the argument v|n in radians

DEGREE(v|n)

Converts the argument v|n (in radians) to degrees

EXP(v|n)

Returns the exponential value of v|n

FRE(v|n)

Returns the first free address:
in the Code Stack (v|n=0)
in the Number Stack (v|n=1)
or in the Text Stack (v|n=2)
or the last free address:
in the Code Stack(v|n=3)
in the Number Stack (v|n=4)
or in the Text Stack (v|n=5).

Note:
- The free space is the difference between the two returned addresses for a given Stack. For instance, FRE(0) returns the first free address not used by your program in the Code Stack. You may then POKE [0..65535] values to the Code Stack from this address on up to FRE(3), that is in the [FRE(0)..FRE(3)] range.
- See the MINOSVERSION compiling directive and appendix #8.

INSTRING(c|t_1 , c|t_2 , v|n)

Searches for text string c|t_2 in c|t_1, starting at position v|n in the first text string

Notes:
- this is a case less search, i.e. “HELLO” and “hello” are the same
- returns the relative position to v|n where text string was first found or 0 if it was not found

INT(v|n)

Returns the largest integer less than or equal to the argument v|n

LEN(c|t)

Returns the length in bytes (number of characters) of c|t

LOG(v|n)

Returns the decimal logarithm of the argument v|n

LN(v|n)

Returns the natural logarithm of the argument v|n

MAX(v|n_1 , v|n_2)

Returns the maximum value of v|n_1 and v|n_2

MIN(v|n_1 , v|n_2)

Returns the minimum value of v|n_1 and v|n_2

NOT(v|n)

Returns the negation of the argument v|n

PEEK(v|n)

Returns the [0..65535] value at address v|n in the Code Stack which has to be in the range [1..FRE(3)]

POWER(v|n_1 , v|n_2)

Returns v|n_1 raised to power of v|n_2

Note:
- is equivalent to v|n_1 ^ v|n_2

RADIAN(v|n)

Converts the argument v|n (in degrees) to radians

RND(v|n)

Returns an integer pseudo-random number in the [0..v|n[range

ROUND(v|n)

Returns the nearest integer to the argument v|n

SGN(v|n)

Returns the sign of the argument v|n, 1 for positive numbers, 0 for 0, and -1 for negative numbers

SIN(v|n)

Returns the sine value of the argument v|n in radians

SQRT(v|n)

SQRT returns the square root of the argument v|n

TAN(v|n)

Returns the tangent value of the argument v|n in radians

TICKS

Returns the system ticks

TICKSPERSEC

Returns the number of ticks per second

Note:

- The ticks per second value depends on the operating system, it is 100 for Palm OS <= 5 devices, if not down- or over-clocked

VAL(c|t)

Returns the numerical value of c|t

TextFunctions:

BIN$(v|n)

Returns the binary-string value of argument v|n

CHAR$(c|t , v|n)

Returns the v|n th character in text string c|t

CHR$(v|n)

Returns a one-character text string with the character corresponding to the ASCII code indicated by argument v|n

DATE$

Returns the current date based on the computer's internal clock as a string in the form "DD/MM/YYYY"

Note:

- As implemented under iziBasic, DATE$ cannot be used for assignment (i.e., to set the system date)

HEX$(v|n)

Returns the hexadecimal-string value of argument v|n

LCASE$(c|t)

Returns c|t converted to lower case

LEFT$(c|t , v|n)

Returns the v|n number of leftmost characters of c|t

LTRIM$(c|t)

Returns c|t after having removed leading blank spaces from c|t

MID$(c|t , v|$n_1$, v|n_2)

Returns the v|n2 part (*length*) of the text string c|t starting from position v|n_1

OCT$(v|n)

Returns the octal-string value of argument v|n

RIGHT$(c|t , v|n)

Returns the v|n number of rightmost characters of c|t

RTRIM$(c|t)

Returns c|t after having removed trailing blank spaces from c|t

SPACE$(v|n)

Returns a text string of blank spaces v|n bytes (characters) long

STR$(v|$n_1$, v|n_2)

Returns a text string giving the representation of the argument v|n_1 with the number v|n_2 of decimals to return

Note for 2nd argument (v|n_2):

- If v|n<0 then return number in exponential notation
- If v|n=0 then return integer part of number

TIME$

Returns the current time based on the computer's internal clock as a string in the form "HH:mm:ss"

Note:

- As implemented under iziBasic, TIME$ cannot be used for assignment (i.e., to set the system time)

TRIM$(c|t)

Returns c|t after having removed leading and trailing blank spaces from c|t

UCASE$(c|t)

Returns c|t converted to upper case

WORD$(c|t , v|n)

Returns the v|n th word in text string c|t

Note:

- words are separated by spaces, tabulations or punctuation marks (comma, period, semi-colon, colon, question mark, exclamation mark)

Defined Constants

EXP	is 2.718281828
FALSE	is 0
MAYBE	is 0.5
PI	is 3.141592654
TRUE	is 1
VERSION	iziBasic version, format is M.m (Major.minor)

Console

iziBasic's console, encapsulates 13 lines of text in which you can PRINT and INPUT some text or numbers. iziBasic will manage the vertical scrolling of these 13 lines.

CLS

Clears the console display screen

Notes:
- CLS does not clear the whole screen, it clears the 13 lines of text
- Would you wish to clear the full screen, you then need to use a BOXFILLED on the area, using the backcolor returned by the COLOR function (see Graphics)

INPUT [c|t ,] c|v

Notes:
- If the first c|t parameter is set (even to an empty string), the input is directly written on the current console text line as it is keyed in (with a possible correction, using the backspace character, being a pen move from middle right to middle left in Graffiti) and the input is memorized when the user keys in a line feed (or carriage return, being a pen move from top right to bottom left in Graffiti) character.
  This is the "standard" old fashion console way of doing an INPUT.
- If the first c|t parameter is omitted, displays a TEXTFIELD (for c) or NUMFIELD (for v) to read a text line or a number and a [Enter] button to validate the entry. Once [Enter] was pressed the input is assigned to c or v. The input field and the [Enter] button are then hidden.

PRINT

Outputs a line feed to the console screen

PRINT c|t [;]

Outputs c|t and a line feed to the console screen unless the facultative [;] argument is set

PRINT v|n_1 [USING v|n_2] [;]

Outputs v|n_1 in the v|n_2 format and a line feed to the console screen unless the facultative [;] argument is set

Notes for USING v|n_2:

- Gives the number of decimals to display
- If v|n<0 then display number in exponential notation
- If v|n=0 then display integer part of number
- If USING v|n is not set, then will display the number in exponential notation (equivalent to set it < 0)

WAIT

Displays the [Enter] button and wait for this [Enter] button to be pressed

TextFunctions:

INKEY$

Reads the status of the keyboard (or a key keyed in in Graffiti) and returns a single keypress, if available

Note:

- Returns an empty text string if no key was pressed. You will then track key pressed within a WHILE / WEND, a DO / LOOP or a REPEAT / UNTIL loop. Example: REPEAT : K$=INKEY$: UNTIL K$<>""

Graphics

Legend:
x is X coordinate and y is Y coordinate, both are v|n

BOX [x_1 , y_1] TO x_2 , y_2
Draws a box from current top-left (x,y) position if first arguments are omitted or from (x_1,y_1) if passed, to the bottom right (x_2,y_2) position

BOXFILLED [x_1 , y_1] TO x_2 , y_2

Draws a filled box with the set COLOR statement from current top-left (x,y) position if first arguments are omitted or from (x_1,y_1) if passed, to the bottom right (x_2,y_2) position

COLOR v|n

Sets the pen color to the v|n value

Note:
- in black & white screen mode, v|n is 0 (white) or 1 (black)
- in other screen modes, v|n = Red x 65536 + Green x 256 + Blue, where Red, Green and Blue are [0..255] gradients

GOTOXY x , y

Places the drawing pen at position (x,y)

Note:
- GOTOXY does not draw a pixel, use PSET for doing so

GPRINT c|t , x , y , [v|n_1 [, v|n_2]]

Displays c|t text at position (x,y), with frontcolor v|n_1 and backcolor v|n_2.
v|n_1 and v|n_2 are facultative parameters, current values for front and back colors are used if these parameters are not defined.

Notes:

- Backcolor can only be set if frontcolor is also set, but frontcolor alone may be defined
-  for v|n_1 and v|n_2 parameters
- The parameters have changed for GPRINT in version 4.2 as from those valid in versions 4.0 and 4.1 (GPRINT c|t , v|n , x , y), so you will have to upgrade your source code accordingly

$ IMAGE v|n, x, y

Puts image v|n at with the top-left corner of the image being at position (x,y)

Note:

- v|n is the image ID which automatically adapts to the color displaying capabilities of the device: high resolution color; low resolution color, gray scale or black & white. Please refer to the IMAGEBUTTON instruction to see the list of available images shipped with iziBasic and to the RESOURCEFILE compiling directive to see how to add your own customized images to your programs.
- ⓘ Warning: there is no check on this image ID number as you can very well add your own customized images to your program. You may also remove or replace this set of images, please refer to the RESOURCEFILE compiling directive for further information.

LINE [x_1 , y_1] TO x_2 , y_2

Draws a line from current (x,y) position if first arguments are omitted or from (x_1,y_1) if passed, to the (x_2,y_2) position

PSET x , y

Draws a pixel at position (x,y)

SCREEN v|n

Sets screen mode:

0 for black & white	3 for 256 colors
1 for 4 grays	4 for 65536 colors
2 for 16 grays	

Notes:

- 
- for devices equipped with Palm OS versions prior to 3.5, screen mode is set to 0 (black & white)
- be careful to set your screen mode before drawing any graphic or GUI object on the screen

$ SETRES v|n

Attempts to set the screen size to 320x320 pixels (high resolution) if v|n = 1 and to 160x160 pixels (standard low resolution) if v|n = 0.

Notes:

- Not all Palm OS APIs support the high resolution mode, so you should set on (when needed) and off (when no more needed) the high resolution mode all along your source code. In other words, the high resolution mode cannot be set once for all in a program!
- Since this statement returns no information, you should first check if the device supports high resolution with the HIGHRES function.

NumFunctions:

COLOR(v|n)

Returns backcolor if v|n = 0 and frontcolor if v|n = 1

COLORRGB(v|n_1 , v|n_2 , v|n_3)

Returns a color given the Red (v|n_1), Blue (v|n_2) and Green (v|n_3) gradients.
Therefore v|n_1, v|n_2 and v|n_3 have to be in the [0..255] range.

$ HIGHRES(v|n)

Attempts to set the screen size to 320x320 pixels (high resolution) if v|n = 1 and to 160x160 pixels (standard low resolution) if v|n = 0.
Returns 1 if the function succeeded, 0 otherwise.

Notes:
- If HIGHRES(1) returns 1, the device has a high resolution screen.
- See the SETRES statement

PGET(x , y)
Returns current color of pixel at position (x,y)

Note:
- 

POSX

Returns the current X position of drawing pen

POSY

Returns the current Y position of drawing pen

SCREENMODE

Returns the current screen mode:

0 for black & white	3 for 256 colors
1 for 4 grays	4 for 65536 colors
2 for 16 grays	

SCREENMODES

Returns the best screen mode available on the device:

0 for black & white devices	3 for 256 colors devices
1 for 4 grays devices	4 for 65536 colors devices
2 for 16 grays devices	

GUI

Legend:

x is X (left) coordinate, y is Y (top) coordinate, w is Width and h is Height,
all of x, y, w and h are v|n
#v|n is the Object ID and is in the range [1..999]

$ ABOUTBOX c|t_1 [+ c|t_2] [+ c|t_3]

When the user clicks on the Application name, opens your customized AboutBox, otherwise displays the default AboutBox (which says that your application was made with iziBasic)

Notes:

- You may pass up to 3 text strings to build an AboutBox with up to 186 characters long (including line feeds)
- This statement is overridden by the MENU statement: when the user clicks on the Application name, your menu will be opened instead of the AboutBox.
- As from version 5.0, the AboutBox can be managed dynamically at runtime and is no more built once at compilation time. This means that you should define your AboutBox after the BEGIN statement to have it taken into account.

ADVICEBOX v|n

Pops up a message window, which has a vertical scroll bar if the text to be displayed requires it, and waits for the user to press the [Done] button.

Notes:

- v|n is a message ID as provided in a resource file, the resource being of type tSTR (String resource, please refer to the RESOURCEFILE compiling directive)
- v|n is to be in the range [1..999]

BUTTON #v|n , c|t , x , y , w , h

Creates and displays a button with c|t label

CHECKBOX #v|n , c|t , 0|1 , x , y , w , h

Creates and displays a checkbox with c|t label and defines if it is initially checked or unchecked (0|1 = unchecked | checked)

$ CLOSEFORM

Closes a custom form and returns to the main form or closes the main form (which is, by default, built and displayed when the program is launched) if no custom form is opened.

Notes:

- Closing the main form does not mean that no screen will be displayed; an empty screen is shown instead. If you build GUI objects at this stage, they will not be shown until you open the main form again. But you may draw some "stuff" (see the Graphics module); this "stuff" will be erased when opening back the main form.
- See the OPENFORM statement.

$ DESTROY #v|n

Hides an object, inactivates it and destroys it

Notes:

- because of a bug in Palm OS prior to version 4.0 (so in versions 3.x), you should destroy an object before using again the same Object ID. To know which version of Palm OS is on the client device, use the GETOSVER$ function and code consequently (i.e. create even empty objects at the program start, then destroy them just before reusing them), or limit your application with the {MINOSVERSION "4.0"} compiling directive.

  example for Palm OS prior to version 4.0:

  ```
  LABEL #1,"Hello World",50,50
  …
  DESTROY #1 : LABEL #1,"Bye, Bye",70,70
  ```

- for later Palm OS versions (version 4.0 and later), iziBasic handles automatically the reuse of an Object ID by first deleting the previous object and then building the new one, so that you do not have to destroy an object before using again the same Object ID.

  example for Palm OS version 4.0 or later:

  ```
  LABEL #1,"Hello World",50,50
  …
  LABEL #1,"Bye, Bye",70,70
  ```

- for maximum compatibility, you might want to always proceed like explained for versions prior to version 4.0, as this way of doing will work for all Palm OS versions starting with version 3.0.
- 

the Handspring Visor devices do not support the DESTROY statement because of what I suspect to be a bug in their Palm OS 3.x special versions.

FIELDCOPY #v|n

Copy the current selection of a NUMFIELD, TEXTFIELD or TEXTFIELD$$ to the clipboard

FIELDCUT #v|n

Copy the current selection of a NUMFIELD, TEXTFIELD or TEXTFIELD$$ to the clipboard and delete the selection from the field

FIELDPASTE #v|n

Replace the current selection in the field, if any, with the contents of the clipboard

FIELDUNDO #v|n

Undo the last change made to a NUMFIELD, TEXTFIELD or TEXTFIELD$$, if any. Changes include backspace, cut and paste.

FLUSHEVENTS [v|n]

According to the facultative [v|n] parameter, offers to flush part or the entire system events queue:

- if no argument was passed or if v|n is set to 0:
  executes pending system events until the events queue is emptied
- if v|n is set to a value in the [1..255] range:
  executes a maximum of v|n system events
  if the events queue is emptied before this v|n number is reached, the control is returned to your application

Notes:

- In the very specific case your source code updates many GUI objects without returning the control to Palm OS with one of the DOEVENTS or WAITEVENT functions, you may get an events queue overflow (resulting in an error message and an application crash). Then insert some FLUSHEVENTS statements in your source code to have the events executed and the events queue emptied.
- Flushing events can be useful in the case you develop an application which allows the user to keep pressing hard buttons and your application has difficulty to manage all its tasks in the time between two events are queued.

$ GRAFFITISHIFT 0|1 , x , y

Sets or unsets the Graffiti Shift indicator at position (x,y)

Notes:
- 0|1 = unset | set the Graffiti Shift indicator
- in the case of unsetting the Graffiti Shift, the x and y coordinates are fake

$ HIDE #v|n

Hides an object and inactivates it, but, unlike DESTROY, does not delete it

Notes:
- You may unhide a hidden object with the SHOW statement
- Prior to Palm OS version 3.2, there was a bug. As a consequence, this function did not set the usable bit of the object attribute data to false, so it could still redraw or respond to the pen.

$ IMAGEBUTTON #v|n_1 , v|n_2 , x , y , w , h

Creates an image button, puts image v|n_2 centered in the button frame

Notes:

- v|n_2 is the image ID with the following images available (which automatically adapts to the color displaying capabilities of the device: high resolution color; low resolution color, gray scale or black & white):

- palm 3.5
- ⓘ Warning: there is no check on this image ID number as you can very well add your own customized images to your program. You may also remove or replace this set of images, please refer to the RESOURCEFILE compiling directive for further information.

KEYBOARD v|n

Pop up the system keyboard if there is a field object with the focus. The field object's text is edited directly. The v|n parameter can take the following values:

0 Alphabetical letters (abc)
1 Numerals (123)
2 Alphabetical letters with accents (Int'l)

LABEL #v|n , c|t , x , y

Creates and displays a label with c|t label.

LISTCHOICE #v|n_1 , c|t_1 | v|n_2 , c|t_2 , x , y , w , h

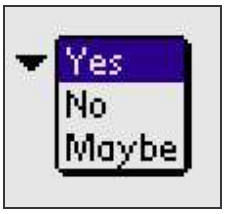
Creates and displays a list choice with initial c|t_1 label or the v|n_2 index of an item in the c|t_2 list of selectable items.
When the user will tick on the LISTCHOICE, a selection of items (c|t_2) will popup and, upon selection, the selected item will replace the initial label.

Notes:

- to set the initial selection, you may either pass its label (c|t_1) or its index (v|n_2) in the list of selectable items
- c|t_2 is the list of selections separated by a ¶ character (which is CHR$(182)) with a maximum of 7 items
  example: "choice #1¶choice #2¶choice #3"
  L$="choice #1"+CHR$(182)+"choice #2"+CHR$(182)+"choice #3"
- c|t_2 may also be a pointer to the A$() array, starting at index A$($n_4$) and stopping at the first empty A$() index, still with a maximum of 7 items
  example: "A$(68)" stops when A$(m)="" with m>68 and m<=74
- the difference between the LISTCHOICE statement and the POPUPCHOICE statement is the way the item selections popup window is presented:

LISTCHOICE

POPUPCHOICE



$ MENU v|n

Loads a menu from a resource file and integrates it to your application.

Notes:

- v|n is a menu ID as provided in a resource file, the resource being of type MBAR (please refer to the RESOURCEFILE compiling directive) in the range [1..999]
- v|n is to be in the range [1..999] to activate the corresponding menu
- If v|n is set to 0, no more menu will be integrated and the default behaviour will be restored (Default or custom AboutBox when clicking on the title)
- This statement overrides the ABOUTBOX statement: when the user clicks on the Application name, your menu will be opened instead of the AboutBox. You should then implement an AboutBox (with MESSAGEBOX or NOTICEBOX) as an answer to a menu item.
- As from version 5.0 of iziBasic, the menu can be managed dynamically at runtime and is no more built once at compilation time. This means that you should define your menu after the BEGIN statement to have it taken into account.

NUMFIELD #v|n , c|t , 0|1 , x , y , w , h

Creates and displays a field to input numeric values, with the initial value set to c|t

Notes:

- the initial value has to be passed as a TextVar, therefore you should convert your v|n to a c|t using the STR$ function
- 0|1 = single line | multiple lines
- only numbers can be keyed in by the user, but the result is returned in a TextVar

$ OPENFORM v|n

Loads the main form or a custom modal form from a resource file and displays it on the screen.

Notes:

- if v|n = 0, the main form is shown. This is useful only in the case it was hidden using the CLOSEFORM instruction.
- Otherwise, v|n is a form ID as provided in a resource file, the resource being of type tFRM (please refer to the RESOURCEFILE compiling directive) in the range [1..999] with the Modal flag set
- If you open a custom form on top of an already opened form, the previous form will be closed before the new one is loaded and displayed. So, there is no need to explicitly close a form before opening a new one, it is only needed if you want to return to the main form.
- All events occurring on objects defined in a custom form that are similar and which follow the same rules (ID in the range [1..999]) to those defined in this GUI module will be captured, except for the LISTCHOICE and POPUPCHOICE ones (Popup).
- ⓘ So, you should avoid using custom forms with Popups (Popups are made of a PopupTrigger,and an attached List). But you may very well add them dynamically in such a form (using the LISTCHOICE and POPUPCHOICE statements), as well as any other object. Just proceed the same way as you do with any object in the main form.
  One additional operation (in 3 steps) will be required for the POPUPCHOICE use which is to add:
  - Step #1. a List resource in your custom form (one List resource only, whatever the number of POPUPCHOICEs you have in your custom form), with ID set to 1012, and unset its usable flag (all other parameters set as you wish, they will be overridden at run time)
  - Step #2. a “fake” PopupTrigger resource, with ID set to 1011 and usable flag unset (all other parameters set as you wish, they will be overridden at run time)
  - Step #3. a Popup resource, linking PopupTrigger 1011 and List 1012

- ⓘ Scrollbars can be included in a custom form, but they need a special treatment too: you should include up to 6 scrollbars in your form, setting their resource ID, starting from 1021 and up to 1026, and unset their usable flag (all other parameters set as you wish, they will be overridden at run time). Then, use the SCROLLBAR statement in your source code to activate them.
- ⓘ Another tricky thing in using OPENFORM has to do with the Palm OS version.
  - ✓ With Palm OS 5: when a MBAR ID is assigned to a tFORM, calling OPENFORM automatically links the menu to the form, as expected.
  - ✓ With Palm OS < 5: when a MBAR ID is assigned to a tFORM, calling OPENFORM does not link the menu to the form.

  This can easily be managed in your iziBasic source code:

```
V$=GETOSVER$
' assign menu 2 to the new form if Palm OS < 5
IF V$--<"5.0" MENU 100
OPENFORM 100
'do whatever is needed with the form
CLOSEFORM
' back to main menu of main form
IF V$--<"5.0" MENU 1
```

- See the CLOSEFORM statement.

POPUPCHOICE #v|n_1 , c|t_1 | v|n_2 , c|t_2 , v|n_3 , x , y , w , h

Creates and displays a popup list choice with initial c|t_1 label or the v|n_2 index of an item in the c|t_2 list of selectable items.
When the user will tick on the POPUPCHOICE, a selection of items (c|t_2) will popup and, upon selection, the selected item will replace the initial label.

Notes:

- to set the initial selection, you may either pass its label (c|t_1) or its index (v|n_2) in the list of selectable items
- c|t_2 is the list of selections separated by a ¶ character (which is CHR$(182)) with no maximum number of items but the number of items you can put in the c|t_2 string (so 32 one character items is the limit)
  example: “choice #1¶choice #2¶choice #3”
  L$=“choice #1”+CHR$(182)+”choice #2”+CHR$(182)+”choice #3”
- c|t_2 may also be a pointer to the A$() array, starting at index A$($n_4$) and stopping at the first empty A$() index, this time with a maximum of 89 items
  example: “A$(68)” stops when A$(m)=”” with m>68 and m<=156
- v|n_3 is the number of items visible in a list, it has to be in the [1..14] range
- see the LISTCHOICE statement’s last note to know what the cosmetic difference is between these two very similar statements

PUSHBUTTON #v|n , c|t , 0|1 , x , y , w , h

Creates and displays a push button with c|t label and defines if it is initially pushed or not (0|1 = not pushed | pushed).

RESTORESCREEN

Restores a screen which was previously saved with the SAVESCREEN statement.

SAVESCREEN

Saves the current screen to be restored later with the RESTORESCREEN statement.

SCROLLBAR #v|n_1 , v|n_2 , x , y , w , h

Creates and displays a scrollbar, putting the scroll car at position v|n_2.

Notes:

- There are 100 positions in the scrollbar, whatever its size, so v|n_2 has to be in the [1..100] range ; you may consider it as a percentage or relative position
- Your application may have up to 6 scrollbars, but you will barely need more than one (usually vertical) or two (usually a vertical one and a horizontal one)
- 

Vertical scrollbars can be used whatever Palm OS version is running the device, but Palm OS 3.5 or later is required for horizontal scrollbars
ⓘ Be careful that iziBasic does not check for the Palm OS version when drawing a horizontal scrollbar, so your code should manage it (see the GETOSVER$ function and the MINOSVERSION compiling directive)

SETFOCUS #v|n

Set the focus to the specified v|n TEXTFIELD or NUMFIELD object.

Note:

- You may use SETFOCUS #0 to unset the focus
- See the GETFOCUS function

SETFONT v|n

Sets text font to be applied in all further objects creations and/or displayings.
The possible values for v|n are:

0	stdFont
1	boldFont
2	largeFont
3	symbolFont
4	symbol11Font
5	symbol7Font
6	ledFont
7	largeBoldFont
8..127	some of the newer devices have additional fonts installed, and even though they are not standard, you may access them
128	iziBasic console low resolution font
129	iziBasic console high resolution font
130	any personalized low resolution font that you put in a resource file
131	any personalized high resolution font that you put in a resource file

Notes:

- Unless forced by the CONSOLEFONT compiling directive, fonts 128 and 129 are only put in your program by iziBasic if they are required (use of INPUT and PRINT statements) and only one of the two will be available at runtime, depending on the screen resolution, low or high, of the device (see the HIGHRES function).
  Because only one is made available at runtime, you may call SETFONT with either of the 128 or 129 values.
- As for your personalized fonts, the same “selection” process according to the device’s screen resolution is applied. Your resources have to be of type NFNT and with the corresponding ID value, 130 or 131 (see the RESOURCEFILE compiling directive).
- Would you wish to only provide a low resolution font to all types of devices, then only provide a font with ID 130 and no font with ID 131. iziBasic will know how to handle it.
- If you only provide a high resolution font, the stdFont will be shown on devices with low resolution display.
- 

for fonts 128,129, 130 and 131

$ SHOW #v|n

Shows a hidden object or redraws a visible object

Note:
- You may hide an object with the HIDE statement

TEXTFIELD #v|n , c|t , 0|1 , x , y , w , h

Creates and displays a field to input some text, with the initial value set to c|t

Notes:
- 0|1 = single line | multiple lines
- see the TEXTFIELD$$ statement in the MegaString chapter

TEXTSELECTOR #v|n , c|t , x , y, w, h

Creates and displays a text selector field, with the initial value set by c|t, to input some text formatted in a way that your program should handle.

Notes:
- When the user ticks on a text selector, it will create an event like a button event and your program should then handle what to do with this event
- The text selector use is especially convenient to call the Color, Date or Time selectors (see COLORSELECT, DATESELECT$ and TIMESELECT$ functions)

$ TITLE c|t

Sets main form title to c|t

Note:
- By default the main form title is set to the program name (see the iziBasic source code skeleton paragraph)

$ UPDATECHOICE #v|n , c|t

Updates the selection of items in the following GUI objects: LISTCHOICE and POPUPCHOICE

Note:
- c|t is the list of selections separated by a ¶ character (which is CHR$(182))

$ UPDATEFIELD #v|n , c|t

Updates to c|t the text of a NUMFIELD or a TEXTFIELD with the c|t value

Note:
- using UPDATEFIELD is a way to avoid deleting (DESTROY) and rebuilding a field from scratch (this was the only way in iziBasic version 1.0)

$ UPDATELABEL #v|n , c|t

Updates to c|t a label's text created with LABEL

Note:
- this is a way to avoid deleting (DESTROY) and rebuilding a LABEL from scratch (this was the only way in iziBasic version 1.0)

$ UPDATEPOS #v|n , x , y

Updates the position of an object to (x,y)

Note:
- To avoid display rendering problems, it is usually wise to HIDE the object, update its position and then SHOW it again

$ UPDATETEXT #v|n , c|t

Updates the text of the following GUI objects: BUTTON, CHECKBOX, LISTCHOICE, POPUPCHOICE, PUSHBUTTON, TEXTSELECTOR

Note:
- this is a way to avoid deleting (DESTROY) and rebuilding an object from scratch (this was the only way in iziBasic version 1.0)

$ UPDATEVALUE #v|n_1 , 0|1 | v|n_2

Updates the value of the following GUI objects:

CHECKBOX	0|1 = not checked | checked
LISTCHOICE	v|n_2 = selected item index
POPUPCHOICE	v|n_2 = selected item index
PUSHBUTTON	0|1 = not pushed | pushed
SCROLLBAR	v|n_2 = position of scroll car in the [1..100] range

Note:
- this is a way to avoid deleting (DESTROY) and rebuilding one object from scratch (this was the only way in iziBasic versions prior to 4.2 for CHECKBOX and PUSHBUTTON)

NumFunctions:

CHECKBOX(#v|n)

Returns 0 if the given checkbox is unchecked and 1 if it is checked

COLORSELECT(v|n)

Opens the Palm OS standard color selector window, and returns the selected color

Notes:
- v|n is the initial proposed color
- 

DOEVENTS

Returns on which object an event occurred (if one occurred)

Return value	Comment	KEYEVENTS required
-24, -23, -22, -21	App Key #4, #3, #2, #1	ON
-13	HotSync Button on Cradle	ON | PARTIAL
-12	Page Down Button	ON | PARTIAL
-11	Page Up Button	ON | PARTIAL
-10	Power On/Off Button	ON | PARTIAL
-1	ExitAppRequest	
0	No event	
#v|n in [1..999]	Object that was clicked (Button, CheckBox, ListChoice, PopupChoice, PushButton, ScrollBar or TextSelector)	
1000	Pen event (get related information from the PENDOWN, PENX and PENY functions)	
1001	Menu event (get related information from the MENUITEM function)	
1002	ABOUTBOX event (for information back to your program)	

Notes:

- You will usually include a DOEVENTS in a DO / LOOP, REPEAT / UNTIL or WHILE / WEND loop and track for the exit of program condition.
  Example:
  ```
  REPEAT : E=DOEVENTS : [Statement] : UNTIL E<0
  ```
- Unless you want to monitor hard keys (which are usually devoted to launching programs, you just have to check that DOEVENTS returns a negative value to exit your program if the KEYEVENT directive is set.

FONTSELECT(v|n)

Opens the Palm OS standard font selector window, and returns the selected font

Note:

- The v|n parameter is the default highlighted font in the dialog box and it has to be one of these three values:
  0 stdFont
  1 boldFont
  7 largeBoldFont

FONTWIDTH(c|t , v|n)

Returns the width in pixels of the c|t text for a given font v|n

Note:
- Please refer to the SETFONT statement as for the possible values for v|n

GETFOCUS

Returns the object ID that has the focus

MENUITEM

Returns the last menu item ID which was selected during the last menu event, 0 if the menu was just opened or -1 if the menu was just closed

Notes:
- The returned ID is the one of the menu item as defined in the MBAR resource
- You should capture the menu event with a DOEVENTS or a WAITEVENT and get the selected menu item just afterwards

MESSAGEBOX(c|t_1 [+ c|t_2] [+ c|t_3] , v|n)

Pops up a message window, with a choice of return buttons given by v|n, and returns the button pressed (1=first , 2=second …).
The message can be defined at runtime (main difference with NOTICEBOX, see below).

Notes:
- You may pass up to 3 text strings to build a MessageBox with up to 186 characters long (including line feeds)
- v|n is the type of MessageBox with the following return buttons:
  - 0= Done
  - 1= OK
  - 2= OK|Cancel
  - 3= Yes|No
  - 4= 1|2
  - 5= 1|2|3
  - 6= 1|2|3|4
  - 7= 1|2|3|4|5
  - 8= 1|2|3|4|5|6
  - 9= 1|2|3|4|5|6|7

NOTICEBOX(v|n)

Pops up a message window, with a choice of return buttons given by v|n, and returns the button pressed (1=first , 2=second …) or 0 if the message resource does not exist. The message has to be defined at design time (main difference with MESSAGEBOX, see above).

Notes:

- v|n is a custom message ID as provided in a resource file, the resource being of type Talt (Alert resource, please refer to the RESOURCEFILE compiling directive)
- v|n is to be in the range [1..999]
- the differences between MESSAGEBOX and NOTICEBOX are that MESSAGEBOX's text is created on the fly but it comes with predefined buttons sets when NOTICEBOX takes a predefined message from a resource file but allowing more customized buttons labels. If you do not know or want to manage a resource file, then consider using MESSAGEBOX

PENDOWN

Returns 1 when pen is down and 0 when pen is up

Note:

- PENDOWN is trapped during a DOLOOP or a WAITEVENT call (return value of pen event is 1000)

PENX

Returns X position of where the pen was last ticked on the screen

Note:

- PENX is trapped during a DOLOOP or a WAITEVENT call (return value of pen event is 1000)

PENY

Returns Y position of where the pen was last ticked on the screen

Note:

- PENY is trapped during a DOLOOP or a WAITEVENT call (return value of pen event is 1000)

PUSHBUTTON(#v|n)

Returns the status of a push button: 0=not pushed or 1=pushed

SCROLLBAR(#v|n)

Returns the position of the scroll car of a scrollbar, in the [1..100] range

SELECTEDCHOICE

Returns the selected item (0=non selected, 1=first , 2=second …) of the last LISTCHOICE or POPUPCHOICE event

Note:

- You should capture the ListChoice or the PopupChoice event with a DOEVENTS or a WAITEVENT and get the selected item just afterwards

WAITEVENT

Returns the same values as DOEVENTS (except for the 0=no event value)

Note:

- be aware that control is not returned to your program until one event occurs, use DOEVENTS if you need to get control back between events.
  In other words,
  A=WAITEVENT
  is equivalent to:
  REPEAT : A=DOEVENTS : UNTIL A<>0

TextFunctions:

DATESELECT$(c|t)

Opens the Palm OS standard date selector window, and returns the selected date

Notes:
- c|t is the default date sent to the Date Selector, format is “DD/MM/YYYY”. Pass an empty text string if you wish to send the current date.
- date format returned is “DD/MM/YYYY” or empty if the user cancelled the input

FIELD$(#v|n)

Retrieves the value stored in a NumField or a TextField

Note:
- to convert the returned value in a NumField to a number, use the VAL function
- see the FIELD$$ function in the MegaString chapter

TIMESELECT$(c|t)

Opens the Palm OS standard time selector window, and returns the selected time

Notes:
- c|t is the default time sent to the Time Selector, format is “HH:mm”. Pass an empty text string if you wish to send the current date.
- time format returned is “HH:mm” or empty if the user cancelled the input
- be careful with this time format as it differs from the “HH:mm:ss” format used in the TIME$ function
- 

Preferences

Palm OS provides a convenient way of storing applications preferences, something similar to the Microsoft Windows registry.
iziBasic gives access to storing and retrieving such preferences. To avoid any risk of deleting other applications preferences, it encapsulates the Preferences database accesses for the current application only.

Legend:
#v|n is the Pref Number and is v|n in the range [1..999]

DELETEPREF #v|n

Deletes a preference from the Palm OS Saved Preferences database

SAVEPREF #v|n , v|n|c|t

Saves a preference, which can be either a numerical or a text string value, to the Palm OS Saved Preferences database

NumFunctions:

LOADPREF(#v|n)

Returns a numerical preference

TextFunctions:

LOADPREF$(#v|n)

Returns a text string preference

Arrays

There are 2 arrays defined in iziBasic: A() and A$().
At runtime, both can address all Numbers and text Strings stacks (see Appendix #8).

A-Z variables are also addressed with A(1)-A(26)
A$-Z$ variables are also addressed with A$(1)-A$(26)
This means, for instance, that A$(2) and B$ are the same thing, A(4) and D are also the same.

DIM A(n)

At design time, DIM A(n) reserves some space in the Numbers stack in addition to the A-Z variables

Notes:
- n>26 and n<=FRE(4) (see the MINOSVERSION compiling directive and appendix #8). You will have to leave some space in the upper stack for all other numerical assignments
- DIM A(n) must be defined before the iziBasic compiler reserves some space in the Numbers stack for its use, so you should place your DIM A(n) at the top of your program, just after the Compiling Directives
- See Appendix #8

DIM A$(n)
At design time, DIM A$(n) reserves some space in the Text stack in addition to the A$-Z$ variables

Notes:
- n>26 and n<=FRE(5) (see the MINOSVERSION compiling directive and appendix #8). You will have to leave some space in the upper stack for all other text assignments
- DIM A$(n) must be defined before the iziBasic compiler reserves some space in the Text stack for its use, so you should place your DIM A$(n) at the top of your program, just after the Compiling Directives
- See Appendix #8

CONST A$(n) = t

Define A$(n) as a text string constant.

Note:
- Doing A$(123)=”Some text” will reserve 1 extra space in the Text stack, when CONST A$(123)=”Some text” will only require no extra space but A$(123) which is always reserved.

CONST A(n) = n

Define A(n) as a numerical constant.

Note:
- Doing A(123)=123 will reserve 1 extra space in the Number stack, when CONST A(123)=123 will only require no extra space but A(123) which is always reserved.

[LET] A$(v|n) = c|t|s [+ c|t|s] [...]

LET assigns the value of the expression after the “=” sign to the variable A$(v|n).
iziBasic supports implied LET statements, meaning that the LET statement is facultative.

Note:
- s can also be a A$(v|n)

[LET] A(v|n) = v|n|f [MathOper v|n|f] [...]

LET assigns the value of the expression after the “=” sign to the variable A(v|n).
iziBasic supports implied LET statements, meaning that the LET statement is facultative.

Notes:
- please see the PARSER compiling directive
- f can also be a A(v|n)

RSORT A , $v|n_1$, $v|n_2$

Reverse sorts array by values from the $v|n_1$ index to the $v|n_2$ one

RSORT A$, v|n_1 , v|n_2

Alphabetical reverse sort of array from the v|n_1 index to the v|n_2 one
The sort is case sensitive

SORT A , v|n_1 , v|n_2

Sorts array by values from the v|n_1 index to the v|n_2 one

SORT A$, v|n_1 , v|n_2

Alphabetical sort of array from the v|n_1 index to the v|n_2 one
The sort is case sensitive

NumFunctions:

MEAN A(v|n_1 , v|n_2)

Returns the average value of array between the v|n_1 index and the v|n_2 one

MIN A(v|n_1 , v|n_2)

Returns the smallest value of array between the v|n_1 index and the v|n_2 one

MAX A(v|n_1 , v|n_2)

Returns the highest value of array between the v|n_1 index and the v|n_2 one

SUM A(v|n_1 , v|n_2)

Returns the sum of all values of array between the v|n_1 index and the v|n_2 one

Files

iziBasic is limited to work on database files (pdb), it does not work on program files (prc). It is even limited to work on specific database files (“DATA”, “Data” and “data” types) in most instructions. This was made on purpose, to avoid any risk of performing dangerous actions on files or deleting programs on the devices.

Legend:
- #v|n is the File Handle and is in the range [0..9]; this means that you can work with up to 10 files simultaneously
- c|t is the name of the database file; do not put the “.pdb” extension in this name

CLOSE #v|n

Closes the file indicated by #v|n which has been previously opened by the OPEN statement, and releases the #v|n handle

Note:
- Raises a FILEERROR if the operation fails

$ COPY c|t_1 , c|t_2 , [c|t_3]

Creates file with name c|t_2 and copies content of file c|t_1 into this new file
If the c|t_3 parameter is passed, sets Creator ID in destination file c|t_2 with a new 4 characters value passed in c|t_3 instead of the Creator ID of c|t_1

Note:
- If file c|t_2 already exists does nothing and raises a FILEERROR

ⓘ Warning: you are allowed to copy databases with any Creator ID, this means that you can copy almost any database file (with “DATA”, “Data” or “data” type) on your device.

INPUT #v|n , v|c

Reads data from an opened file with OPEN and moves to next record

Note:
- Raises a FILEERROR if the operation fails
- All data, even numerical values, is stored in text string format in the file (using WRITE #). So you may write a number and read it back in a text string

$ KILL c|t

Deletes the file specified by **c|t**

ⓘ Warning: you are allowed to delete databases with any Creator ID, this means that you can delete almost any database file (with “DATA”, “Data” or “data” type) on your device.

Note:
- Raises a FILEERROR if the operation fails

OPEN c|t FOR INPUT|OUTPUT|APPEND|RANDOM AS #v|n

Makes file c|t available for sequential input, sequential output and reserves the #v|n handle for all read and write accesses to this file
The FOR part of the instruction passes the file open mode:
- INPUT sequential reading (with INPUT #), starting at first record
- OUTPUT sequential writing (with PRINT #), erasing all content of file if any when opening it or creating it if does not exist
- APPEND sequential writing, beginning at current end of file (see EOF function), creates the file if it does not exist
- RANDOM random-access reading and writing (see SEEK statement), starting at first record, creates the file if it does not exist

Notes:
- Raises a FILEERROR if the operation fails
- To avoid any risk of deleting important files on your Palm device, the OUTPUT, APPEND and RANDOM file modes only work with databases having a “LDIB” Creator ID if the SECUREFILES compiling directive is set to ON
- Be careful: if you erase iziBasic from your device, which has “LDIB” as a Creator ID, the standard Palm OS deletion mode will also erase all databases having a “LDIB” Creator ID
- Would you wish to work on external files anyway (meaning not having a “LDIB” Creator ID), you may:
  - either set the SECUREFILES compiling directive to OFF
  - or do it in 3 steps by using the COPY instruction with “LDIB” as Creator ID for the destination file before using the OPEN instruction with this destination file. After you finish working on this file, CLOSE it. Then you may KILL the original file and COPY the destination back to the original one.

  ⓘ Warning in the case you work on such databases, be careful that applications using categories may not work correctly afterwards! This is due to how Palm OS manages categories. In the case of the Memo Pad for instance, my own tests have shown that the APPEND mode works fine when the RANDOM mode would give unpredictable results…

PRINT #v|n , v|n|c|t

Writes data to file and moves to next record

Note:
- Raises a FILEERROR if the operation fails
- In RANDOM mode, erases the current record and replaces it with the new one, unless the file pointer is at the end of file (see EOF function)
- All data, even numerical values, is stored in text string format in the file. So you may write a number and read it back (using INPUT #) in a text string

$ RENAME c|t_1 , c|t_2

Renames file from name c|t_1 to name c|t_2

Note:
- Raises a FILEERROR if the operation fails

ⓘ Warning: you are allowed to rename databases with any Creator ID, this means that you can rename almost any database file (with “DATA”, “Data” or “data” type) on your device.

$ RUN c|t_1 [, c|t_2]

Exits from the current program and launches program c|t_1 with a facultative c|t_2 string text parameter to pass to the new program

Note:
- if the program passed in parameter c|t_1 does not exist, remains in the current program and you might then want to handle the error with FILEERROR in the lines following the RUN instruction…
- the c|t_2 parameter is passed to the new program which may retrieve it with the RUN$ function

$ SEEK #v|n_1 , v|n_2

Sets file position to the given v|n_2 record

Note:
- SEEK is available for files opened in INPUT or RANDOM modes

NumFunctions:

EOF(#v|n)

Returns if End Of File is reached (1=true / 0=false)

Notes:

- The current record number is the next record to read or write. For instance, if you just read the 3rd record (with the INPUT #v|n, v|c statement) out of 4 records, EOF will return true.
- As a consequence, to parse all records of a database you should code something like this:

```
WHILE F=0
  F=EOF(#1) : INPUT #1,A$ : PRINT A$
WEND
```

and not like this which will skip the last record:

```
WHILE F=0
  INPUT #1,A$ : F=EOF(#1) : PRINT A$
WEND
```

FILEERROR

Returns if last file operation generated an error (1=true / 0=false)

FILEEXISTS(c|t)

Returns 1 = file exists or 0 = file does not exist

LOC(#v|n)

Returns LOcation in File = current record number

Note:

- The current record number is the next record to read or write. For instance, if you just read the 3rd record (with the INPUT #v|n, v|c statement), LOC will return 4.

LOF(#v|n)

Returns Length Of File = number of records

TextFunctions:

FINDFIRST$(c|$t_1$, c|t_2)

Returns the name of the first file found which complies with the given parameters (see notes) or empty text string if none found

- c|t_1 is a 4 characters Type (application, database…) of files to search for, pass empty text string to scan all types
- c|t_2 is a 4 characters CreatorID of files to search for, pass empty text string to scan all Creator IDs

FINDFIRST$ works in conjunction with FINDNEXT$. You initiate a search for files with FINDFIRST$ and then search for all given files with FINDNEXT$ in a loop.

Notes:

- scan for the next files complying with the search criteria using the FINDNEXT$ function

FINDNEXT$(c|$t_1$, c|t_2)

Returns the name of next file found which complies with the given parameters (see notes) or empty text string if none was found anymore

FINDFIRST$ works in conjunction with FINDNEXT$. You initiate a search for files with FINDFIRST$ and then search for all given files with FINDNEXT$ in a loop.

Notes:

- c|t_1 is Type and c|t_2 is CreatorID of database to search for
- be careful to pass the same parameters as for the FINDFIRST$ function, otherwise you will get unexpected results

Example of use:

```
' Scan for all iziBasic database files
F$=FINDFIRST$("DATA","LDIB")
WHILE F$<>""
  PRINT F$
  F$=FINDNEXT$("DATA","LDIB")
WEND
```

RUN$

Retrieves a string text parameter passed to the program.

Note:

- see the RUN statement

InfraRed Beaming

BEAMFILE c|t

Send a c|t file to another device by IrDA beaming. This file can be received by the launcher on another Palm OS platform device. It can also be accepted on a PC over IrDA.

Notes:

- Raises a FILEERROR if the operation failed, that is if the transfer did not succeed. This is the case for instance when the database is copy protected.
- Be careful that if the transfer succeeded, it does not mean that the target device kept the data!

Sound

PLAYWAVE v|n_1 , v|n_2 , v|n_3

Plays a wave music with the following parameters:

- v|n_1 is a wave sound ID that should be included in your resource file (“WAVE” type, see RESOURCEFILE compiling directive)
- v|n_2 is the sound volume, which must be in the [0..32768] scale
- v|n_3 is the synchronization mode. If set to 0, plays sound asynchronously, if set to 1 plays sound synchronously

Note:

-  and only if sound feature is present on the device. If not a file error is returned which can be trapped with the FILEERROR function

SOUND v|n_1 , v|n_2 , v|n_3

Plays a single sound with the following parameters:

- v|n_1 is the sound frequency in Hertz
- v|n_2 is the sound volume, which must be in the [0..64] scale
- v|n_3 is the sound duration in milliseconds

System

$ CLIPBOARDGET c|v

Gets the content of the clipboard and stores it in a variable

$ CLIPBOARDPUT c|t|v|n

Put the content of c|t|v|n in the clipboard

NumFunctions:

$ BATTERYINFO(v|n)

According to the parameter passed in v|n, returns:
- 0: 1 if the device is being charged, 0 otherwise
- 1: the current percentage of the battery charge
- 2: the current voltage in volts
- 3: the warning voltage in volts
- 4: the critical voltage in volts
- any other value: returns 0

TextFunctions:

GETOSVER$

Returns the Palm OS version of the device, in the “M.m” format (Major.minor)

$ HOTSYNCINFO$(v|n)

According to the parameter passed in v|n, returns:
- 0: the last hotsync date
- 1: the last hotsync time
- 2: the hotsync user ID
- any other value: returns an empty text string

Note:
- returns empty text strings if the device was never synchronized

PP Code Segment and ARMlet Calls – “PP applets”

Palm Pascal onboard Compiler (nicknamed “PP”) is a great, free and very fast compiler available on the Palm platform. iziBasic itself is made with this compiler which can be found here: http://www.ppcompiler.org/.
PP has this great features of being able to handle:

1. multiple DragonBall code segments
2. “ARMlets” for new devices equipped with ARM processors and Palm OS 5

iziBasic, by construction, cannot handle direct Palm API calls.
So, if for any reason, you need to build very quick routines or to access directly the Palm APIs, you might want to include some PP segments or ARMlets in your iziBasic projects. These code segments and ARMlets are easily added in a resource file (see RESOURCEFILE compiling directive) or on top of an iziBasic compiled program during a PP compilation.

Advices:

- give a look to the iBHelloPP sample program which show a very simple implementation of a PP code segment call and of a PP ARMlet from an iziBasic program
- another more complex sample program, CPDBdemo, shows how to interact with an external library, a very smart one in this case which eases the work on databases
- Khertan (http://www.khertan.net/) from France provides further explanations and useful samples of “PP applets” (320x480 high resolution mode access, 5-way nav button, Address Book access…).

TextFunctions:

$ CALLPP$(v|n , [c|t])

Calls a PP code segment and returns a text string passed by the PP code segment to the iziBasic program

- v|n is the code segment handle and has to be in the [100..999] range (the [0..99] range is reserved for iziBasic)
- [c|t]: pass a facultative text parameter to the PP segment

Note:

- returns a file operation error (1=true / 0=false) which can be read by the FILEERROR function. This can be used to know if the call succeeded

The PP source code skeleton for a segment called by the CALLPP$ function has to be the following:

```
{$code appl,XXXX,code,nnn}
program YourProgramName;
type iBasFunType=function(S:string):string;
var iBasCallPP:iBasFunType;

type
// Insert here any type that your source code requires

const
// Insert here any const that your source code requires

// WARNING: you may insert some global variables here
// Just be careful that they do not use more than a 256 bytes
// size, so as not to override this 256 bytes buffer allowed
// by iziBasic for PP applets global variables

// Insert here any function or procedure that your
// source code requires

// You may rename the CallPP function if you like.
// Any value that you will put in the return string of
// this function will be passed to your iziBasic code.

function CallPP(S:string):string;
begin
 // Insert here some Pascal source code
end;

begin
// Also rename the CallPP function in next line if you
// did it above
 iBasCallPP:=CallPP;
end.
```

Notes:

- everything that is written in **black** is mandatory, in **blue** are comments and facultative definitions, in **green** are parameters to adapt
- **XXXX** is the CreatorID set with the CREATORID compiling directive in your iziBasic source code
- **nnn** is the code segment handle, as called by the CALLPP$ function, please remember that it has to be in the [100..999] range
- replace **YourProgramName** by the program name you defined in your iziBasic source code

$ CALLPPARM$(v|n , [c|t])

Calls a PP ARMlet and returns a text string passed by the PP ARMlet to the iziBasic program

- v|n is the ARMlet handle and has to be in the [100..999] range (the [0..99] range is reserved for iziBasic)
- [c|t]: pass a facultative text parameter to the PP ARMlet

Note:

- returns a file operation error (1=true / 0=false) which can be read by the FILEERROR function. This can be used to know if the call succeeded.
  For instance,if the target device cannot run ARMlets a file operation error will be returned (and you can cross check with the GETOSVER$ function that Palm OS version is 5 or not).

The PP source code skeleton for an ARMlet called by the CALLPPARM$ function has to be the following:

```
{$armlet appl,XXXX,armp,nnn}
program YourProgramName;

// Insert here the string variable passed to the ARMlet by
// the CALLPPARM$ function. Any value that you will put in
// this variable will be returned to your iziBasic code.
// You may change the name of this string variable.
var iBasStr:string;

// Insert here any type, const, var, function or procedure
// that your source code requires

begin
// Insert here any PP source code
end.
```

Notes:

- everything that is written in **black** is mandatory, in **blue** are comments and facultative definitions, in **green** are parameters to adapt
- **XXXX** is the CreatorID set with the CREATORID compiling directive in your iziBasic source code
- **nnn** is the ARMlet handle, as called by the CALLPPARM$ function, please remember that it has to be in the [100..999] range
- replace **YourProgramName** by the program name you defined in your iziBasic source code

MegaString

Text strings are limited to 62 characters in iziBasic. But, in some cases, it is needed to access wider chunks of data, especially for database files accesses. For example, you might want to read a complete memo from the memo database, or a DOC record.
It is a common habit to limit records size in databases to 4 Kbytes (4096 bytes).

So, iziBasic comes with ONE so called “MegaString”, which is 4 Kbytes long and that you can use for these purposes, or also to store any data of your wish.

Note: even though you can put some character anywhere in the MegaString (see the PUTCHAR$$ and PUTSTRING$$ statements), the MegaString is considered as a CHR$(0) ended text string (like all other text strings).

ⓘ Warning: the MegaString is not passed with CHAIN, so, if you need it in the new code segment, you should save its content to a temporary file before chaining and reload it from this file in the new code segment.

CLEAR$$

Clears the MegaString

Note:
- Fills all characters of the MegaString with CHR$(0) characters

GETFIELD$$ #v|n

Retrieves the value stored in a field (NUMFIELD, TEXTFIELD or TEXTFIELD$$) to the MegaString

Note:
- see the FIELD$ function in the GUI chapter

INPUT$$ #v|n_1 [,v|n_2]

Input of one file record into MegaString (refer to the Files paragraph)

Note:
- if the facultative v|n_2 parameter is set in the [1..4096] range, a block of v|n2 characters long record will be read, whatever CHR$(0) ending text string characters might be found within the given range.

$ PRINT$$ #v|n_1 [,v|n_2]

Print the content of MegaString to a file record (refer to the Files paragraph)

Notes:

- All characters of the MegaString before first occurrence of a CHR$(0) character are written to the file
- if the facultative v|n_2 parameter is set in the [1..4096] range, a block of v|n2 characters long record will be written, whatever CHR$(0) ending text string characters might be found within the given range.

$ PUTCHAR$$ s|t , v|n

Put one character s|t at index v|n of MegaString, v|n is in the range [1..4096]

$ PUTSTRING$$ s|t , v|n

Put one text string s|t starting at index v|n of MegaString, v|n is in the range [1..4096]

TEXTFIELD$$ #v|n_1 , v|n_2 , x , y , w , h

Creates and displays a field to input some text, up to v|n_2 characters, with the initial value set to the current content of the MegaString.

Notes:

- valid values for v|n_2 are in the [1..4096] range
- see the TEXTFIELD statement in the GUI chapter

NumFunctions:

LEN$$

Returns the length of the MegaString, in other words returns position of the first occurrence of CHR$(0) minus 1.

Note:

- in the case you assigned the v|n_2 parameter in the INPUT$$ or PRINT$$ statements, LEN$$ might not return the real length of your MegaString as there might be some CHR$(0) characters within the v|n_2 range.

TextFunctions:

GETCHAR$$(v|n)

Returns the v|n th character of the MegaString

GETSTRING$$(v|n_1 , v|n_2)

Returns a v|n_2 long text string from the MegaString, starting at position v|n_1

Note:

- If a CHR$(0) character was found, the returned text string will be less than v|n_2 long as it is truncated to all characters found before the CHR$(0) one