iziBasic v6.0

November 4, 2005

Index

Appendixes

What is iziBasic?

Notice: in French "*izi*" is pronounced like "*easy*" in English

iziBasic stands for easy Basic for Palm. It targets all kinds of developers and should be a very good tool for newbee programmers. Skilled programmers will also find in iziBasic a tool to develop very quickly and easily various types of software.

iziBasic is a high level development compiler which builds Stand-alone applications. The great thing is that it does all of that directly on-board of your Palm OS based device. This Stand-alone application builder is very convenient for those wishing to distribute their creations made with iziBasic in a ready to run software.

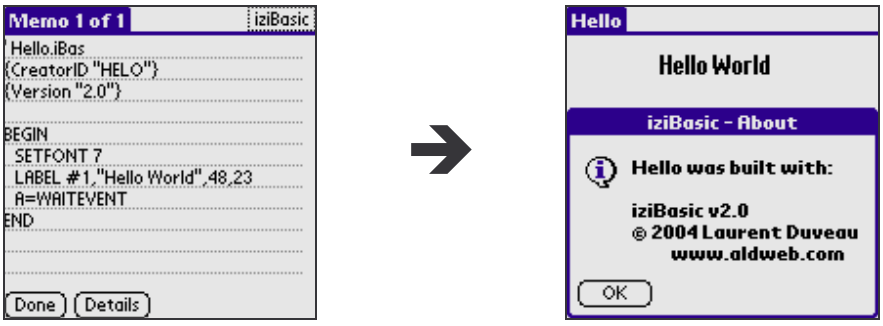
Source codes are easily written using:

- either the Memo Pad application which is shipped with all Palm OS devices,
- or an on-board third party DOC editor of your choice.

iziBasic reads the source codes, compiles them and builds the applications. The use of the Memo Pad is very convenient as you do not need to install any additional editor software to iziBasic. The 4096 characters limit of a memo is not an issue as iziBasic is able to link (or “chain”) together up to 10 memos to build a bigger application. But, for your convenience, you may as well use a DOC editor which overpasses this 4096 characters limit.

As its name also states it, iziBasic uses the BASIC high level and very easy to learn development language, a customized subset of it to be precise. You will discover how easy and quick it is to develop software with iziBasic when the more common development tools available on the Palm OS platform usually require pretty good development skills.

Easiness of the Basic language and Palm hosting are not gained against execution speed of compiled programs. According to my Bench2 analysis (http://www.aldweb.com/articles.php?lng=en&pg=24), iziBasic is one of the Palm hosted tools which generate the fastest programs. It is in 6th position, behind the PP, OnBoardC and QuartusForth (which are compilers), PLua and my LaFac-HELP (which are runtime based like iziBasic). And iziBasic is the fastest on board Basic dialect.



Contact Information

- ✓ World Wide Web main download Site : http://www.aldweb.com
- ✓ Author e-mail : info@aldweb.com

iziBasic is a shareware.

The limitations of the trial version are very light, so as not to stop you from using iziBasic if you like it and cannot afford to buy it.
For instance, there is no time limit and you can very well use iziBasic without any time restriction.
But you are encouraged to support this shareware and buy it.
The limitations are:

- A nag screen to remind you to buy the full version
- Some functionalities are not activated (a few instructions and statements)
- The About box in your programs tells that your software was made with iziBasic

To get a full version of iziBasic, please refer to the iziBasic.txt file that was shipped together with this software or look for iziBasic on my web site (http://www.aldweb.com) and follow instructions.
The cost of iziBasic is just as little as $25.

When you register, you receive a full version of iziBasic that you just need to install on top of this current trial version.

Thanks for purchasing iziBasic.

iziBasic users group & resources

Upon the request of many iziBasic early adopters, I have set up a forum on my web site (**http://www.aldweb.com**) to give you an opportunity to exchange tricks, ask for help & support and share whatever else you would like with the growing community of iziBasic developers.
I will be on the iziBasic forum very often too. So, as to have most people benefit from good ideas, upgrades features requests, beta versions… please use this iziBasic sharing forum rather than sending me e-mails. Then, everybody will get the benefit of your input ☺

Other resources have been created to help you with the iziBasic use: a Yahoo! User Group, a Wiki (both thanks to Montalvo from Mexico), explanations and samples about the so called “PP applets” (thanks to Khertan from France). All links to these resources can be found on this web page:
http://www.aldweb.com/articles.php?lng=en&pg=6085.

How to install iziBasic?

There is nothing special to say here.

iziBasic is a PRC file that is installed like any other Palm file using HotSync.

So, extract **iziBasic_trial.PRC** (or **iziBasic_full.PRC** if you bought the full version) from the ZIP archive file.

Double-click on it and the Palm install tool will popup.
iziBasic.PRC will be transferred to your Palm device next time you synchronize your Palm with your PC with HotSync.

Please uninstall any previously installed version of iziBasic before installing this one.

- ✓ **Minimum Palm OS requirement for iziBasic is version 3.0 with 2 MB of RAM**
- ✓ **iziBasic is Palm OS version 5 and version 6 compliant**

How to use iziBasic?

I'll be very quick in these explanations as iziBasic behaves like all Palm OS based applications, so it is very intuitive to use. ☺

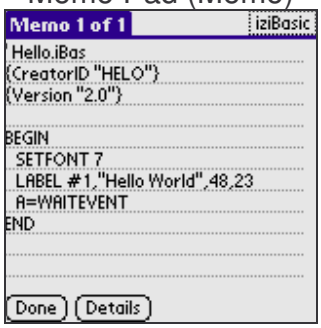
Using the Palm built in Memo Pad or a DOC editor to write programs:

iziBasic searches for the source codes in the Memo Pad database (MemoDB.pdb) and among the DOC files.

On the right side, you see an example of the classical *Hello World* program written in the customized Basic understood by iziBasic.

All Palm OS based devices are provided with an inbuilt Memo Pad application. This lets you write single programs up to 4096 characters. Single programs can be linked (or "chained") to build bigger programs.

Memo Pad (Memo)



Using a third party DOC editor (see Appendix #3) is just as easy. You will then be able to overpass the 4096 characters limit of the Memo Pad (some third party extended Memo Pad software do it also but they are not supported by iziBasic).

PIAF (DOC Editor)

ⓘ iziBasic does not recognize the DOC compressed format, only the uncompressed format.

Once your programs are written, you want to run them. This is when iziBasic comes to be useful.

Using iziBasic to compile source code:

iziBasic lists in this popup list all source codes found in your device.
The glyph character in front of the source code name tells if the file is:
- a DOC file □
- a Memo §

If you tick on the iziBasic title, you open the About box from which you can access the Options box

As its name states it, the Build it button launches the compilation of the source code selected in the top-middle popup list

The Editor button lets you launch your favorite editor which you defined in the Options box (accessed through the About box)

Main screen display used to display information about compilation

If checked, the Verbose Compiler option will display all processed source code lines when compiling (very useful for debugging!)

If compilation was successful, you may directly run your program from iziBasic by pressing the Run it button (otherwise, this button is hidden)

Key shortcuts are defined to launch different controls in iziBasic's interface:

- ✓ A opens the About box
- ✓ B press the [Build it] button
- ✓ E press the [Editor] button
- ✓ O in the About box or the Options box, press the [OK] button
- ✓ P in the About box, open the Options box
- ✓ R press the [Run it] button
- ✓ V check or uncheck the [Verbose] checkbox

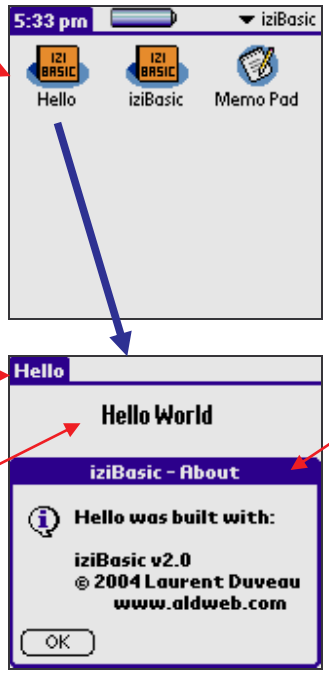
Notes:

- small and capital letters are valid as key shorcuts
- no need to press the Palm shortcut key to activate these shortcuts, just press or digit the adequate key in Graffiti

Running the application:

Once your application was created, it appears as all other Palm OS applications in the launcher

You may now run it…



Trick: you now may want to use RsrcEdit (*free publicity!*) to change this ugly icon and to personalize your application with your own designed icon

The title displayed is the one of the program you built

The Hello World label was created

For this screenshot, I clicked on the title. As a consequence, the default About box was launched

This is the end of this quick tour. iziBasic is just that easy to use. ☺

Maybe consider the ViziBasic add-on?

ViziBasic stands for Visual easy Basic for Palm. It is an add-on application that I especially made for empowering the development of iziBasic projects.

ViziBasic is a Visual editor for iziBasic. In other words, it is a Rapid Application Development tool (so called RAD), a "What You See Is What You Get" (WYSIWYG) form designer, and the perfect companion to the iziBasic compiler.

Further description of ViziBasic features is to be found in Appendix #6.

iziBasic source code skeleton

iziBasic requires you to put these very few lines to detect and compile a minimum source code:

```
' YourProgramName.ibas
{CREATORID "XXXX"}

BEGIN
END
```

Notes:

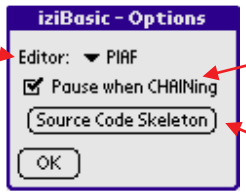
- replace **YourProgramName** by the name you want for your program. This comment line, with a program name ended by a ".ibas" extension) has to be the very first one of your source code for iziBasic to detect that this is an iziBasic source code
- replace **XXXX** by the Palm Creator ID of your program

But, you might rather want to use the Source Code Skeleton wizard which is provided by iziBasic to ease this skeleton setup (see below)

iziBasic Options window

The Options box is accessed through the About box

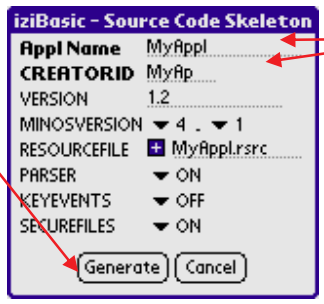
You select here your favorite editor (Memo Pad, PIAF, QED, SiEd, SrcEdit or ViziBasic) to be launched from the Editor button in iziBasic's main window



If checked, compilation pauses between source codes when chained (see CHAIN statement)

The Source Code Skeleton button opens a wizard to generate the basics of a source code

When you click on the Generate button, a source code will be created, either in the Memo Pad format or in the DOC format according to the selected editor in the Options box



Application Name and Creator ID are mandatory parameters (in bold)
All other parameters are facultative (explanations to be found in the Compiling directives ahead)

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.

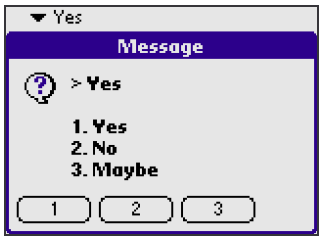
LISTCHOICE #v|n_1 , c|t_1 | v|n_2 , c|t_2 , x , y , w , h

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.

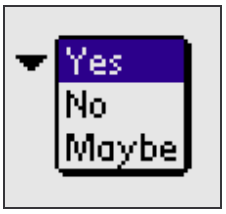
Notes:

- 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

Compiler errors:

The iziBasic compiler is a one pass compiler which processes several checks but some controls are not made. For instance:

- IF THEN [ELSE] END IF without END IF is not checked
- WHILE WEND without WEND is not checked
- FOR I=1 TO 10 : FOR I=1 TO 20 : NEXT : NEXT is not checked (use of the same Number variable in 2 overlapped loops)
- …

Of course, at runtime you will then get unexpected results!

CHAIN Stack Overflow

You are trying to chain more than 10 Memo files.

Advice: group small source codes together rather than splitting them.

Code Stack Overflow

Your code size has over passed FRE(3) records (see the MINOSVERSION compiling directive).

Advice: split your source code in several files and link them with the CHAIN instruction.

DIM defined too late

The compiler has already started to reserve some Numbers or Text values.

Advice: put your DIM statement at the top of your source code, right after the Compiling Directives.

Jump Stack Overflow

Your Jump Stack has over passed its size: 255 records.

Advice: decrease the number of labels and of GOSUB / GOTO instructions.

Number Stack Overflow

Your Number Stack size has over passed FRE(4) records (see the MINOSVERSION compiling directive).

Advice: use the CONST instruction for values that are fixed at the beginning of the program to decrease the Stack.

Notes:

- the Number Stack is of FRE(4) records but the iziBasic compiler uses a variable number of records according to your source code requirements
- the 26 spaces required by the A-Z variables are also reserved by iziBasic in the Number Stack

Single Statement too long

One Statement was over 62 characters. You will have to split it into 2 lines or 2 statements (separated by a ":" character).

Syntax error

There is a syntax error in the last statement processed by the compiler.

Advice: make sure that none of your labels starts with a reserved keyword. For example: CloseMyForm: or GoToMySubRoutine: are not good when MyFormClose and JumpToMySubRoutine as well as _CloseMyForm and _GoToMySubRoutine are valid.

Text Stack Overflow

Your Text Stack size has over passed FRE(5) records (see the MINOSVERSION compiling directive).
Advice: use the CONST instruction for values that are fixed at the beginning of the program to decrease the Stack.

Note:

- the Text Stack is of FRE(5) records but the iziBasic compiler uses a variable number of records according to your source code requirements
- the 26 spaces required by the A$-Z$ variables are also reserved by iziBasic in the Text Stack

Appendix #1: Tutorial - my first iziBasic program

In this tutorial, we will build a very simple first iziBasic program.
This program will display a simple text field and a button. When pressing the button, a popup window will appear showing what the user wrote in the text field.
This is a pretty simple objective for a program, but you will be so happy when you will run it for the first time! ☺

Here is the full source code for this program:

```
' MyFirstPgm.ibas
{CREATORID "MyFP"}

BEGIN
   LABEL #1,"Input some text:",10,40
   TEXTFIELD #2,"",0,85,40,65,14
   BUTTON #3,"Show my Text",40,70,80,14
   REPEAT
      E=DOEVENTS
      IF E=3 THEN
         T$="You wrote:"+CHR$(10)+FIELD$(#2)
         M=MESSAGEBOX(T$,0)
      ENDIF
   UNTIL E=-1
   M=MESSAGEBOX("OK, bye bye then!",0)
END
```

Let's now explain each single line of this source code:

```
' MyFirstPgm.ibas
```

This comment line (because starting with a ' comment sign) is required when the source code is written in the inbuilt Memo Pad, for iziBasic to figure out that this memo is an iziBasic source code memo. In this case as well as in the case of a DOC source code, the passed name in the comment is used as the program name.

```
{CREATORID "MyFP"}
```

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. In our case, we have chosen "MyFP" as a Creator ID.

```
BEGIN
```

We indicate iziBasic that our main routine starts here. In other words, the program's entry point is here.

```
   LABEL #1,"Input some text:",10,40
   TEXTFIELD #2,"",0,85,40,65,14
   BUTTON #3,"Show my Text",40,70,80,14
```

We define here 3 objects to put on the screen: a label, an empty textfield and a button. As from these definitions, our main screen will look like this:

We now have to set up how our program will react to user input on this screen objects. We will therefore set up a loop to capture events (like a button press in our sample source code) and have our program do something accordingly.

```
   REPEAT
```

We enter in the events management loop.

```
      E=DOEVENTS
```

If any event occurs, it is tracked in the E variable. In our sample program, E can only take 3 values:

-1	the user pressed the Home button, asking to leave our program (this is a pity! How could you want to exit from such a great program?)
0	no event occurred
3	an event was captured for object #3. As this object is a button, this was a button press

```
      IF E=3 THEN
         T$="You wrote:"+CHR$(10)+FIELD$(#2)
         M=MESSAGEBOX(T$,0)
      ENDIF
```

When E is equal to 3, it means that our “Show my Text” button was pressed. We then wanted to popup a window showing the text that was keyed in the fieldtext.
This is what we do, storing the value in the T$ variable, and then launching a messagebox with this T$ variable. The CHR$(10) part of the T$ definition inserts a line feed between the “You wrote:” text string and the content of the fieldtext captured thanks to the FIELD$(#2) function.
The returned value in M from the messagebox is not used, but it could in some more complex application…
Here is a screenshot of the result of this IF loop management:


```
   UNTIL E=-1
```

This is the end of our events capturing loop. As long as E is different from -1, we shall loop back to the REPEAT statement. If E is equal to -1, it means that the user pressed the Home button on the silkscreen and asked to leave from this great application. As we like very much our user, we will obey and let him exit!

```
   M=MESSAGEBOX("OK, bye bye then!",0)
```

Since we are very polite, we added this fancy messagebox to say good bye to our user.


```
END
```

We then exit from our application, and go back to Palm’s launcher.

Appendix #2: Sample iziBasic source codes

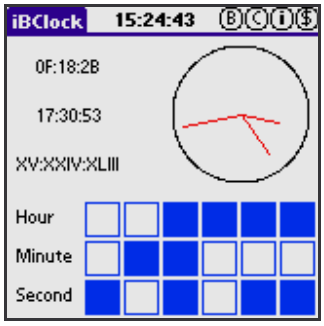
In the iziBasic ZIP distribution file, you will find a directory called Examples.
In this Examples directory are offered full source codes for 20 sample applications.
These applications, for most of them, are complete and operational software that are offered as freeware to all lucky owners of Palm devices. You will find the most interesting ones (iBClock, Matches, NekoCat, Numerus) in many popular download websites.
Reading and working on these source codes will help you to quickly get a full understanding of the iziBasic capabilities as I believe that learning by the example is the best way to learn programming.

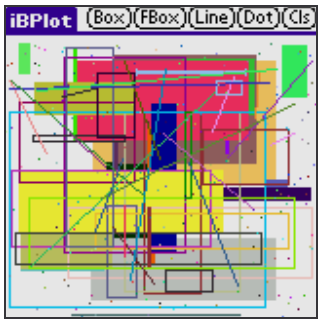
Therefore, I highly invite you to give a deep reading to these sample source codes and to run these sample programs.

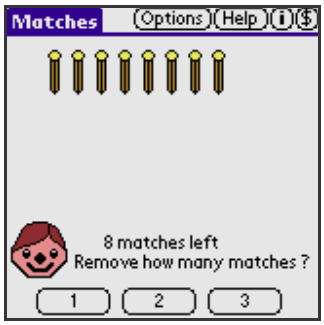
Note: some programs require the full iziBasic to be compiled; the trial version will not compile them. But, I provide a compiled and running version for all sample programs.

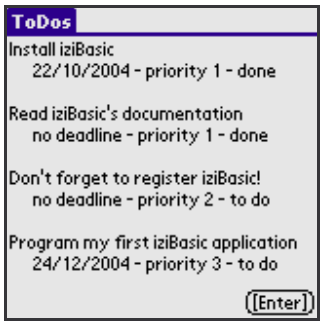
- Source codes which can be compiled with trial version:
  Bench1, Bench2, iBcHello, iBcMatches, iBcNumerus, iBHello, MemoTitles, ToDos
- Source codes which can only be compiled with full version:
  CPDBdemo, iBAddress, iBChristmas, iBClock, iBDim2, iBHelloPP, iBPlot, Matches, NekoCat, Numerus, tinyBasic & Memo2tinyBas

To give you an idea of the power of iziBasic, here are snapshots for 6 of the 20 sample applications:









Appendix #3: PIAF, QED, SiEd and SrcEdit DOC editors

The most sophisticated Palm hosted compilers (PP and OnBoardC for instance) rely on a DOC editor to write the source codes. iziBasic can read both the Memo Pad format or the DOC format.
Using a DOC editor is very convenient in most cases because DOC editors provide great editing features and they overpass the 4096 characters limit of the Memo Pad.

There are two types of DOC formats: a compressed one and an uncompressed one (which was the original one).
ⓘ iziBasic does not recognize the DOC compressed format, only the uncompressed format.

Below, you will find a non exhaustive reference list of five Palm hosted DOC editors (listed by alphabetical order, not by my personal order of preference… even though I am very much linked to PIAF!) which are flavoured by the developers' community and that can be directly launched by iziBasic (other DOC editors could be launched by iziBasic too, do not hesitate to ask).

PIAF

PIAF is the dedicated DOC editor for the Palm Pascal (PP) onboard compiler.

ⓘ With the help of Philippe Guillot, PP's and PIAF's developer, I have upgraded the source code of PIAF to have it handle color coding for iziBasic's keywords and launch iziBasic's compiler. With PIAF, iziBasic can also receive which source code to compile, start compiling automatically (without needing to press the [Build it] button) and send in return to PIAF the line where a compilation error was found so that PIAF can open the source code directly on the given line.

Note: iziBasic itself was developed in PP using PIAF, directly in my Palm Tungsten C. As iziBasic can also work with "PP applets" for extending its capabilities (see the CALLPP$ and CALLPPARM$ functions), you might want to consider PIAF first as your iziBasic source code editor.

URL: http://www.ppcompiler.org/ Cost: free, copyleft source code

QED

QED is a well established DOC editor, well known by the developers as it was one of the first (if not the very first one) Palm hosted DOC editors, and it proved to be very convenient to use for developers.

URL: http://qland.de/qed/ Cost: shareware ($12.95)

SiEd

SiEd is the newest of the DOC editors in this list. It has a very convenient functionality: it can save a file in the DOC format in the device's main memory and in a text format on an external memory card.

URL: http://www.benroe.com/sied/ Cost: free, GPL source code

SrcEdit

SrcEdit is the dedicated DOC editor for the OnboardC compiler. It is of course very well adapted to the developers needs.

ⓘ With SrcEdit, iziBasic can receive which source code to compile, start compiling automatically (without needing to press the [Build it] button) and send in return to SrcEdit the line where a compilation error was found so that SrcEdit can open the source code directly on the given line.

ⓘ John, SrcEdit's main developer, is working on upgrading SrcEdit to handle color coding for iziBasic's and other languages' keywords (with my help for providing the iziBasic keywords file).

Note: SrcEdit is a real alternative to the "native" PIAF offer to use as your iziBasic source code editor.

URL: http://onboardc.sourceforge.net/ Cost: free, GPL source code

Appendix #4: BIRD, Icon Manager, RsrcEdit resources editors

Palm executables and databases are made of resources that are fully managed by Palm OS, like: code, icons, bitmap images, forms and their objects, message windows, help texts, records…

iziBasic manages the building of code resources, together with a few default resources (icons, bitmaps, some message windows).
iziBasic also offers to integrate specific resources (icons, bitmaps, “PP applets”, menus, alert windows, advice texts…), using the RESOURCEFILE compiling directive.
You may very well, in many cases, do not need them, except for applications icons I presume, as you will for sure want to change iziBasic’s default icon by a personalized one for your software.
But, would you wish to customize your software and get the benefit of all possibilities given by iziBasic, then a third party software will be required to build, edit and modify these resources.

Below, you will find a non exhaustive reference list of three Palm hosted resources editors (listed by alphabetical order, not by my personal order of preference… even though I am very much linked to BIRD!) which are flavoured by the developers’ community.

BIRD

BIRD is a wonderful multipurpose resources toolbox, just like RsrcEdit its ancestor. It has been developed by the prolific developer of PP and PIAF. But, unlike RsrcEdit, it is a brand new tool which takes into account all newer needs (long file names display, high resolution bitmaps and icons…).

ⓘ BIRD only works in devices equipped with Palm OS 5, not with OS 4 or earlier versions of Palm OS.

URL: http://www.ppcompiler.org/ Cost: free, copyleft source code

Icon Manager

Icon Manager, as its name states it, is dedicated to editing icons and bitmaps. If you do not plan to work on other resources but your applications icons or want to be able to design icons in a very handy way, Icon Manager is a good tool.

URL: http://www.palmgear.com/index.cfm?fuseaction=software.showsoftware&prodid=47054
Cost: free

Quartus RsrcEdit

RsrcEdit is a wonderful multipurpose resources toolbox, the very first one made available, developed by the genius people of Individeo (those who also developed OnBoardC and SrcEdit at the time), then passed to Quartus.
RsrcEdit has been unique, a must have tool for years, and is commonly used by all onboard developers or software handymen. Unfortunately RsrcEdit has not evolved at all since 2001. The newer BIRD resources editor provides all required enhancements (like high resolution bitmap editing) that are badly lacking in RsrcEdit.

URL: http://www.quartus.net/products/rsrcedit/ Cost: shareware ($15.00)

ⓘ I also provide a way to customize RsrcEdit (using RsrcEdit itself to do it!) to show long files names (that appeared on Palm devices after the last release of SrcEdit in 2001) on my website.
URL: http://www.aldweb.com/articles.php?lng=en&pg=4990

Appendix #5: Tutorial – building and linking resources

Using Custom Icons within iziBasic Applications
By Mike Featherstone

Introduction

The purpose of this document is to illustrate the manner in which custom or 'user defined' icons may be generated and used in place of the iziBasic icons included by default in all iziBasic generated applications.

A number of tools are available to help in an exercise such as this but for the purposes of this example I have used the following applications (parts of which you will see in the accompanying screen shots):

PIAF 1.10 - Available as detailed in the iziBasic user manual (appendix #3)
BIRD - Available as detailed in the iziBasic user manual (appendix #4)
iziBasic itself (of course).

(Using PIAF and iziBasic together provides a very good integrated solution for developing BASIC applications on the PalmOS device.)

How to Generate and Use Icons

For the purpose of this example, I'm starting with the following simple iziBasic program:

```
{CREATORID "MdF7"}
{VERSION "1"}

Repeat
 a=waitevent
until a=-1
```

It does absolutely nothing except terminate when requested but is a good enough foundation for the purposes of this exercise.

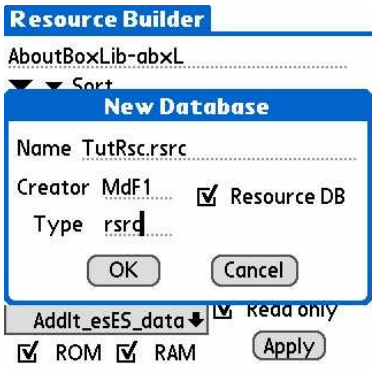
Compiled with iziBasic, the program (Tut) looks like this in the launcher window of my T5:

This is OK but you can see that once you have built a few iziBasic applications (I have 2 here) the display begins to get confusing with no iconic way of distinguishing one application from another. The solution is to design your own icons for each application and incorporate them into your programs at compilation.

This is actually a surprisingly simple process!

Step 1 – The resource file

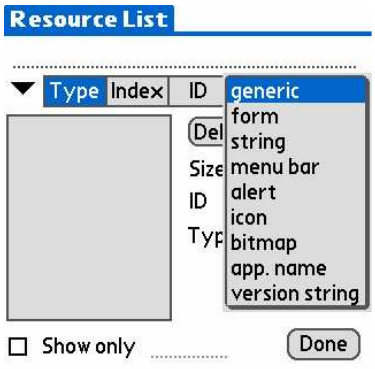
The first step is to create a resource file to contain your new icon. Below is an example, demonstrating how BIRD would be used for this task and the data required – Name, Creator and Type:



(This is not a lesson in using BIRD so I'm not going to detail all of the steps required to complete each stage of this process with this particular application.)

Step 2 – The Resource

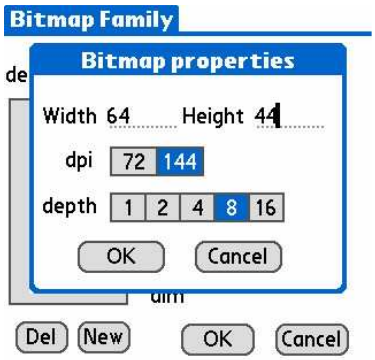
Once you have created your file, you can then create your new resource item. A number of different types are possible (as illustrated) but in this case, we require an icon:



If you started this exercise with an empty resource file, the chances are that you've ended up with your Icon resource having the correct Type and ID assigned from the start. In order to work as a program icon, the Type must be set to **tAIB** and the ID must be **1000**.

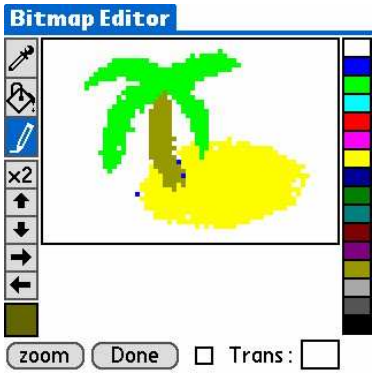
Step 3 – The Icons

Now that you have a suitable icon resource, it may be populated with bitmaps:



An individual icon resource will normally comprise a number of individual bitmaps. These will each have either a different resolution or colour depth so as to correspond to the different types of display available on PalmOS devices. Each device will use the one most appropriate to its display configuration. Normally (I'm told) a low resolution icon is sized 32x22 pixels and a high resolution one is double that at 64x44 pixels (as illustrated) but there seems to be no hard and fast rule about this.

(To state the obvious, the icons themselves are not provided by any of these applications – you will need to construct your own.)



On a practical note, I have found that creating an Icon resource comprising a single high resolution bitmap results in the icon not being displayed centrally. The only way I have found to correct this is to ensure that a low resolution icon is included within the same resource. Furthermore, to ensure that your high resolution icon is centred correctly, the low resolution icon should be exactly half its size i.e. half the height and half the width.

I've no idea why this is!

Step 4 – IziBasic

Now that you have a resource file that you have populated with suitable icons for your application, you need to incorporate it into your iziBasic source code using the RESOURCEFILE compiler directive as follows:

```
{CREATORID "MdF7"}
{VERSION "1"}
{RESOURCEFILE "TutRsc.rsrc"}

Repeat
 a=waitevent
until a=-1
```

A simple recompilation:

and the result is a program that is indicated in your chosen launcher using the icon you designed:

Step 5 – The Other Icon

As well as the ‘large’ icon generally used on the main Launcher page, an application also contains a smaller icon that is used elsewhere (a good example would be where you have your launcher set up to display programs as a list rather than as icons).

These Icons may be created by exactly the same process as described above except that the resource ID for the small Icons should be **1001** and the size of the icons is nominally limited to 9x9 pixels for low resolution devices and 18x18 pixels for high resolution devices. (In practice, I find that 15x9 and therefore 30x18 make optimal use of the space available). As with the lager icons, a low resolution icon of half the pixel width is required to ensure that a high resolution icon is centralised correctly when used.

Mike Featherstone
7/1/2005

Appendix #6: ViziBasic, the visual editor add-on to iziBasic

What is ViziBasic?

ViziBasic stands for Visual easy Basic for Palm. It is an add-on application that I especially made for easing the development of iziBasic projects requiring GUI components (well, most of the Palm applications that you will develop should need that!).

ViziBasic is a Visual editor for iziBasic.
ViziBasic is an application dedicated to:

- design simple GUI applications, with forms and objects in a form
- have objects perform specific actions thanks to attached iziBasic source code to each object, source code that you write to customize these actions
- generate and write the iziBasic source code to some Palm DOC format file that can then be edited like any other iziBasic source code for further enhancing your application
- thanks to its integration with the iziBasic compiler (as its name states it!), compile directly the generated iziBasic source code for immediate result

And all of this is done directly on board of your Palm device.

So, ViziBasic is a Rapid Application Development tool (so called RAD), a "What You See Is What You Get" (WYSIWYG) form designer, and the perfect companion to the iziBasic compiler.
ViziBasic is then especially useful for quickly prototyping an application and, since the source code is written in the standard Palm DOC format, it can be the base for further enhancements or more complex coding.

ViziBasic does not come with iziBasic and it is quite useless without it.
You have to purchase the iziBasic compiler separately.
ViziBasic requires a full version of iziBasic to compile the source codes it generates.
But, in the case you do not own a full version of iziBasic, you may still play around with its user interface and generate the related source codes to help you make up your mind before purchasing it.

Last but not least, ViziBasic itself is coded with iziBasic ☺

URL: http://www.aldweb.com/ Cost: $15

<u>An example of designing a form with iziBasic and ViziBasic</u>

In the following example, we design a simple form with the classical “Hello World” message stored in a label.

In the standard iziBasic use, you input all of your source code manually:

We write all of our source code, following iziBasic’s syntax

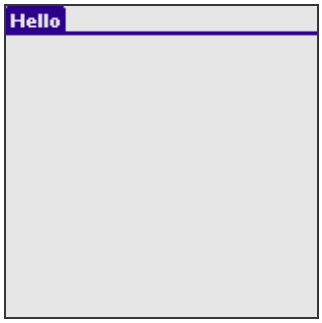
Compiling this source code is required to see the result

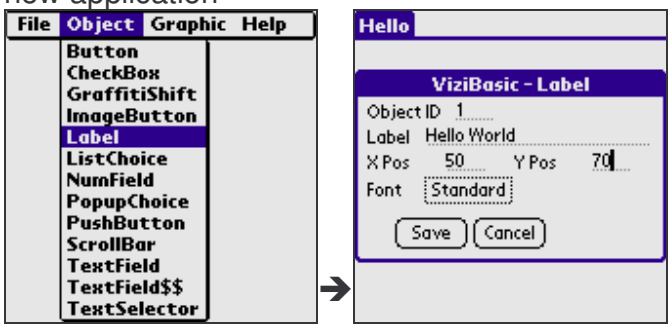
Were we to want to change the label’s position, we would need to go back in the source code, then compile again…

With ViziBasic, you may do the same, but in a visual way:

First, we define the main characteristics of our new application

Result is immediately shown on the screen (title)





Then, we add a label and a wizard window helps us define all required characteristics for such an object

Result is again immediately shown on the screen

To change the label’s position, we just need to select it to go back to the previous window

Appendix #7: BASIC statements & functions reference

Statements & Functions	Type	iziBasic module
ABOUTBOX	Statement	GUI
ABS	NumFunction	Core
ACOS	NumFunction	Core
ADVICEBOX	Statement	GUI
AND	Math Op	
APPEND	Statement	Files
AS	Statement	Files
ASC	NumFunction	Core
ASIN	NumFunction	Core
ATAN	NumFunction	Core
BATTERYINFO	NumFunction	System
BEAMFILE	NumFunction	IR Beaming
BEEP	Statement	Core
BEGIN	Statement	Core
BIN$	TextFunction	Core
BOX	Statement	Graphics
BOXFILLED	Statement	Graphics
BOXFILLEDTO	Statement	Graphics
BOXTO	Statement	Graphics
BREAK	Statement	Core
BUTTON	Statement	GUI
CALL	Statement	Core
CALLPP$	TextFunction	PP applet
CALLPPARM$	TextFunction	PP applet
CASE	Statement	Core
CASE ELSE	Statement	Core
CHAIN	Statement	Core
CHAR$	TextFunction	Core
CHECKBOX	Statement / NumFunction	GUI
CHR$	TextFunction	Core
CLEAR	Statement	Core
CLEAR$$	Statement	MegaString
CLIPBOARDGET	Statement	System
CLIPBOARDPUT	Statement	System
CLOSE	Statement	Files
CLOSEFORM	Statement	GUI
CLS	Statement	Console
COLOR	Statement / NumFunction	Graphics
COLORRGB	NumFunction	Graphics
COLORSELECT	NumFunction	Graphics
CONST	Statement	Core / Arrays

COPY	Statement	Files
COS	NumFunction	Core
DATE$	TextFunction	Core
DATESELECT$	TextFunction	GUI
DEC	NumFunction	Core
DEGREE	NumFunction	Core
DELETEPREF	Statement	Preferences
DESTROY	Statement	GUI
DIM	Statement	Core / Arrays
DO	Statement	Core
DOEVENTS	NumFunction	GUI
DOWNTO	Statement	GUI
ELSE	Statement	Core
END	Statement	Core
END IF	Statement	Core
END SELECT	Statement	Core
EOF	NumFunction	Files
EXP	NumFunction / Constant	Core / Constant
FALSE	Constant	Constant
FIELD$	TextFunction	GUI
FIELDCOPY	Statement	GUI
FIELDCUT	Statement	GUI
FIELDPASTE	Statement	GUI
FIELDUNDO	Statement	GUI
FILEERROR	NumFunction	Files
FILEEXISTS	NumFunction	Files
FINDFIRST$	TextFunction	Files
FINDNEXT$	TextFunction	Files
FLUSHEVENTS	Statement	GUI
FONTSELECT	NumFunction	GUI
FONTWIDTH	NumFunction	GUI
FOR	Statement	Core
FRE	NumFunction	Core
GETCHAR$$	Statement	MegaString
GETFIELD$$	Statement	MegaString
GETFOCUS	NumFunction	GUI
GETOSVER$	TextFunction	System
GOSUB	Statement	Core
GOTO	Statement	Core
GOTOXY	Statement	Graphics
GPRINT	Statement	Graphics
GRAFFITISHIFT	Statement	GUI
GETSTRING$$	Statement	MegaString
HEX$	TextFunction	Core
HIDE	Statement	GUI
HIGHRES	NumFunction	Graphics
HOTSYNCINFO$	TextFunction	System

IF	Statement	Core
IMAGE	Statement	Graphics
IMAGEBUTTON	Statement	GUI
INC	Statement	Core
INKEY$	TextFunction	Console
INPUT	Statement	Console / Files
INPUT$$	Statement	MegaString
INSTRING	NumFunction	Core
INT	NumFunction	Core
KEYBOARD	Statement	GUI
KILL	Statement	Files
LABEL	Statement	GUI
LCASE$	TextFunction	Core
LEFT$	TextFunction	Core
LEN	NumFunction	Core
LEN$$	NumFunction	MegaString
LET	Statement	Core / Arrays
LINE	Statement	Graphics
LINETO	Statement	Graphics
LISTCHOICE	Statement	GUI
LN	NumFunction	Core
LOADPREF	NumFunction	Preferences
LOADPREF$	TextFunction	Preferences
LOC	NumFunction	Files
LOF	NumFunction	Files
LOG	NumFunction	Core
LOOP	Statement	Core
LTRIM$	TextFunction	Core
MAX	NumFunction	Core / Arrays
MAYBE	Constant	Constant
MEAN	NumFunction	Arrays
MENU	Statement	GUI
MENUITEM	NumFunction	GUI
MESSAGEBOX	NumFunction	GUI
MID$	TextFunction	Core
MIN	NumFunction	Core / Arrays
MOD	Math Op	
NEXT	Statement	Core
NOT	NumFunction	Core
NOTICEBOX	NumFunction	GUI
NUMFIELD	NumFunction	GUI
OCT$	TextFunction	Core
OPEN	Statement	Files
OPENFORM	Statement	GUI
OR	Math Op	
OUTPUT	Statement	Files
PEEK	Statement	Core

PENDOWN	NumFunction	GUI
PENX	NumFunction	GUI
PENY	NumFunction	GUI
PGET	NumFunction	Graphics
PI	Constant	Constant
PLAYWAVE	Statement	Sound
POKE	Statement	Core
POP	Statement	Core
POPUPCHOICE	Statement	GUI
POSX	NumFunction	Graphics
POSY	NumFunction	Graphics
POWER	NumFunction	Core
PRINT	Statement	Console / Files
PRINT$$	Statement	MegaString
PSET	Statement	Graphics
PUSH	Statement	Core
PUSHBUTTON	Statement	GUI
PUTCHAR$$	Statement	MegaString
PUTSTRING$$	Statement	MegaString
RADIAN	NumFunction	Core
RANDOM	NumFunction	Core
REM	Statement	Core
RENAME	Statement	Files
REPEAT	Statement	Core
RESTORESCREEN	Statement	GUI
RETURN	Statement	Core
RIGHT$	TextFunction	Core
RND	NumFunction	Core
ROUND	NumFunction	Core
RSORT	Statement	Arrays
RTRIM$	TextFunction	Core
RUN	Statement	Files
RUN$	TextFunction	Files
SAVEPREF	Statement	Preferences
SAVESCREEN	Statement	GUI
SCREEN	Statement	Graphics
SCREENMODE	NumFunction	Graphics
SCREENMODES	NumFunction	Graphics
SCROLLBAR	Statement / NumFunction	GUI
SEEK	Statement	Files
SELECT CASE	Statement	Core
SELECTEDCHOICE	NumFunction	GUI
SETFOCUS	Statement	GUI
SETFONT	Statement	GUI
SETRES	Statement	Graphics
SGN	NumFunction	Core
SHOW	Statement	GUI

SIN	NumFunction	Core
SLEEP	Statement	Core
SORT	Statement	Arrays
SOUND	Statement	Sound
SPACE$	TextFunction	Core
SQRT	NumFunction	Core
STEP	Statement	Core
STR$	TextFunction	Core
SUM	Statement	Arrays
SWAP	Statement	Core
TAN	NumFunction	Core
TEXTFIELD	Statement	GUI
TEXTFIELD$$	Statement	MegaString
TEXTSELECTOR	Statement	GUI
THEN	Statement	Core
TICKS	NumFunction	Core
TICKSPERSEC	NumFunction	Core
TIME$	TextFunction	Core
TIMESELECT$	TextFunction	GUI
TITLE	Statement	GUI
TO	Statement	Core
TRIM$	TextFunction	Core
TRUE	Constant	Constants
UCASE$	TextFunction	Core
UNTIL	Statement	Core
UPDATECHOICE	Statement	GUI
UPDATEFIELD	Statement	GUI
UPDATELABEL	Statement	GUI
UPDATEPOS	Statement	GUI
UPDATETEXT	Statement	GUI
UPDATEVALUE	Statement	GUI
USING	Statement	Core
VAL	NumFunction	Core
VERSION	NumFunction	Constants
WAIT	Statement	Core
WAITEVENT	NumFunction	GUI
WEND	Statement	Core
WHILE	Statement	Core
WORD$	TextFunction	Core
XOR	Math Op	

Appendix #8: FAQ

Calculations precision

As written in this user manual, numbers in iziBasic are 32 bit float IEEE 754 compatible. For most people, this does not mean much! But, it does have an impact on numbers precision, just like for any other float numbers system. Here is a sample source code to illustrate this point:

```
BEGIN
  A=17.05*100       you would expect A to store 1705 (1.7050000e03) as a value
  PRINT A           displays 1.7049999e03 which is the value really stored in A
  A$=STR$(A,2)
  PRINT A$          displays 1705.00 as STR$ has a precision optimiser
  B=INT(A)          B receives INT(1.7049999e03)=1704
  PRINT B           displays 1.7040000e03, not 1.7050000e03
END
```

Complex aggregate of calculations

As explained earlier in this documentation, v|n or c|t cannot be an aggregate of calculations in [LET] statements.

With some other Basic compilers or interpreters you could do (with an example):

```
E=5*MAX(C,MIN(A,B))
```

With iziBasic, you should work this way:

```
E=MIN(A,B) : E=5*MAX(C,E)
```

So far, I do not intend to change this behaviour.

Complex tests in conditional statements

All conditional statements require a “v|n TestOper v|n” or a “c|t TestOper c|t” structure.

With some other Basic compilers or interpreters you could do (with an example):

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

With iziBasic, you should work this way:

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

or this way too:

```
{PARSER ON} C=(3*COS(B)+5)<MAX(A,B) {PARSER OFF}
IF C=TRUE PRINT “OK”
```

Upgrading the compiler in this area is in my roadmap (see appendix #9) even though I personally like very much the current way of how it is implemented, because I believe that it structures better the source code.

Boolean test on strings

Currently, in its math parser, iziBasic does not allow to work with Boolean tests on strings. The math parser expects every single argument to be of a number type (Number, Defined Constant, NumVar or NumFunction).
So, the way to do a Boolean test in iziBasic is to use the “c|t TestOper c|t” structure in a conditional statement.

With some other Basic compilers or interpreters you could do (with an example):

```
A=(A$<”test”)
```

With iziBasic, you should work this way:

```
A=0 : IF A$<”test” LET A=1
```

So far, I do not intend to change this behaviour.

Memo on Palm Tungsten T5 (and other newer palmOne devices)

Memos can override the 4 Kb limit in the Palm Tungsten T5. As from my knowledge, this is the first device to do so. Nevertheless, iziBasic assumes that a memo is not bigger than 4 Kb and it will not read source code above this limit.
Would your memo be bigger, compilation will then end up in a non expected way (usually with a syntax error).
So, if you are a happy owner of a Palm Tungsten T5 and wish to write long iziBasic programs, please consider using a DOC editor (see appendix #3).

Serial communication

The serial communications is not a function offered by iziBasic (yet?). So, something like OPEN “COM1:” FOR INPUT AS #1 does not work at the present time.

Do you plan to add VFS file management, High resolution support for 320x480 screens, 5 way navigator handling, socket capabilities, etc…?

By construction, iziBasic uses a virtual machine to run the compiled programs.
The positive point of this way of doing is that all of the hard and low level programming is made by me (!) and you may rely on iziBasic's high level syntax to write small and yet powerful source codes which, once compiled, get all of the hard stuff done by the virtual machine. But, the side effect of this search for easiness is that iziBasic cannot be exhaustive and cannot let you handle direct calls to the whole set of the numerous (and always evolving) Palm OS or third parties APIs.

Overall, my development strategy for iziBasic is the following:

- I will stick to the standard Palm OS APIs as provided by PalmSource (the company which develops Palm OS and which is no more the same company as PalmOne which develops some devices)
- I will not add to iziBasic things specific to one device or a few devices (like the 320x480 screen, 5 way navigator…) until they are made available in the standard Palm OS API set

Nevertheless, iziBasic is not shut up in its current set of features (which, I believe, is already very wide!). Indeed, all that is not provided in a "native" way by iziBasic can be developed thanks to the opening of the so called "PP applets" (see the PP Code Segment and ARMlets Calls – "PP applets" paragraph in this manual).
Then, you may get the benefits of both iziBasic and PP:

1. develop quickly and easily with iziBasic's high level syntax most of your application
2. add to it all of the specific stuff you want with "PP applets"

So, and I really ask it as a favour given the number of requests I receive, please consider the "PP applet" option before asking me for something in this area.
If you chose iziBasic because of its easiness (thank you, I am very happy then as this has always been my main objective for developing it!) and if you do not feel like getting into the internals of Palm OS programming with the Pascal language of PP, then I would recommend you to reconsider your development project. Think about it, there are good reasons… This being said, I never close the door to good ideas: many of the enhancements of the previous and current releases of iziBasic were made thanks to its devoted users! And I plan to add many other features in the future (see Appendix #9).

Note: The Pascal syntax of the PP compiler is very close to the one of the C language which is widely used. I have personally not (yet?) tried to see how these "PP applets" principle can extend to other development languages, like the C language, but I have no doubt that we could, for instance, have "C applets" too. So, if some people have the knowledge of programming in C for Palm OS, I would be interested in receiving feedback and help.

Appendix #9: iziBasic Memory structure

iziBasic works with 4 stacks: a code Stack, a Numbers Stack, a Text Stack and a Jump Stack (but this last one is used internally and you do not have access to it).
Their different sizes are explained in the MINOSVERSION compiling directive and can be retrieved with the FRE function.

The Code stack holds the compiled code which is executed. It can be read with the PEEK function and you can even put some values in it with the POKE statement.

You have more influence on the Numbers and the Text Stacks and you can interact in a wider way with these stacks. This appendix provides explanations for that.

Memory organization for the Numbers stack:

Physical address	**Use**
FRE(4)+6	Last stack address
↑	Unused space
FRE(1)+6	First free stack address
n+1..FRE(1)+5	Addresses reserved and set at compilation time
33..n	Space for the A(27..n-6) array and/or other Numbers variables reserved with the DIM statement
7..32	A to Z variables
1..6	6 records reserved by system

Memory organization for the Text stack:

Physical address	**Use**
FRE(5)+2	Last stack address
↑	Unused space
FRE(2)+2	First free stack address
n+1..FRE(2)+1	Addresses reserved and set at compilation time
29..n	Space for the A$(27..n-2) array and/or other Text variables reserved with the DIM statement
3..28	A$ to Z$ variables
1..2	2 records reserved by system

Retrieving Stack values

At runtime, you may use the A() and A$() arrays in a very convenient way to read and/or write Number or Text values at any address but the records reserved by system.
The index of A() and A$() start at the address of A and A$, meaning that A(1)=A, so A(1) points to the real "physical" address 7.
So, you just have to provide the index in the range [1..FRE(4)] for A() and in the range [1..FRE(5)] for A$().

Physical address	A() index address	A$() index address
FRE(4)+6	A(FRE(4))	
↑	↑	
FRE(5)+2	↑	A$(FRE(5))
↑	↑	↑
8	A(2)=B	↑
7	A(1)=A	A$(5)=E$
6		A$(4)=D$
5		A$(3)=C$
4		A$(2)=B$
3		A$(1)=A$
2		
1		

Stacking values in a given order

The use of the DIM statement allows you to stack up variables and arrays the way you want. Let's illustrate it with two examples:

Case #1:

```
DIM A(32)
DIM %MyVar1%
DIM %MyVar2%
```

A() index address	Use
27..32	A(27..32)
33	%MyVar1%=A(33)
34	%MyVar2%=A(34)

Case #2:

```
DIM %MyVar1%
DIM A(32)
DIM %MyVar2%
```

A() index address	Use
27	%MyVar1%=A(27)
28..32	A(28..32)
33	%MyVar2%=A(33)

Appendix #10: Developing an iziBasic project on a Windows/Linux/Mac computer

Even though one of iziBasic's main reasons of being is to do all of the programming work and the compiling directly on board of the Palm device, some people find it convenient to write their source code in a PC or a Mac and test the result in the Palm Emulator or Palm Simulator.

Here below you will find three examples of ways to code and test an iziBasic project on a Windows PC, a Linux device or a Mac computer:

- ✓ the first explanations are to be run on a Windows equipped PC using the freeware PsPad editor and MakeDoc conversion software.
  - *Explanations for a full integration with the Palm emulator and simulator are included as this is my own main hardware platform*

- ✓ the second explanations are to be run on a Windows equipped PC using the freeware ConTEXT editor and MakeDoc conversion software.
  - *This is an alternative to the PsPad editor use, just to show how easy it is to switch from one editor to another one*

- ✓ The third explanations will be for any device that can run a Sun Microsystems' Java Runtime Environment (Java virtual machine), so any Windows/Linux/Mac equipped computer will make it, using the iziEditor especially developed by Khertan for iziBasic as its name states it.
  - *I do not know very well the Linux Operating System and even less the Mac one. As a consequence, the explanations will be quite limited. The Palm emulator is also available on these Operating Systems (but, as far as I know, the Palm simulator only runs on a Windows PC).*

Other tools are available either on Windows, Linux or Mac to build similar development environments.

For instance, if you think that the parameterization of the PsPad example here below to build a very smooth and nice development environment on your Windows PC is a little bit too complex and that you do not care about syntax color highlighting or full integration with the Palm Emulator, you might consider using tools like TL-PDB (http://www.freewarepalm.com/utilities/tl-pdb.shtml). You might also want to consider the iziEditor option!

So, please consider this appendix only for what it is: an example of how to develop an iziBasic project on another device but a Palm.

Developing an iziBasic project on a Windows PC using the PsPad editor

Software & Files needed:

PsPad
PSPad editor is a great freeware programmer editor which works with almost any programming environments, providing highlighted syntax in code.
URL: http://www.pspad.com/en/ Cost: free
Version available at the time I am writing this appendix: 4.3.3 (2089) - 6/29/2005

iziBasic.INI / iziBasic.BAT files
I provide these files which are the configuration files for PsPad to provide the iziBasic highlighted syntax and the automation of the DOC building.
iziBasic.INI is to be found in the iziBasic ZIP distribution file, under the " *Editors Highlighters\PsPad* " folder.
iziBasic.BAT is to be found in the iziBasic ZIP distribution file, under the " *Editors Highlighters* " folder.

MakeDoc
MakeDoc converts a plain text file to a DOC formatted file that can then be transferred to a Palm device and read by any DOC reader.
URL: http://nature.berkeley.edu/~alyons/palm_cybercafe.html Cost: free

Palm OS Emulator and/or Palm OS Simulator
The Palm OS Emulator emulates a Palm device, up to OS 4.x. It is an "emulator" because it needs a real copy of an OS ROM to run.
The Palm OS Simulator simulates a Palm device, for OS 5.x and OS 6.x. It is a "simulator" because the OS is compiled to run natively with a PC processor.
URL: http://www.palmos.com/dev/dl/dl_tools/ Cost: free

Note: if you choose other directories but those provided in the following instructions, you will have to customize all further explanations accordingly.

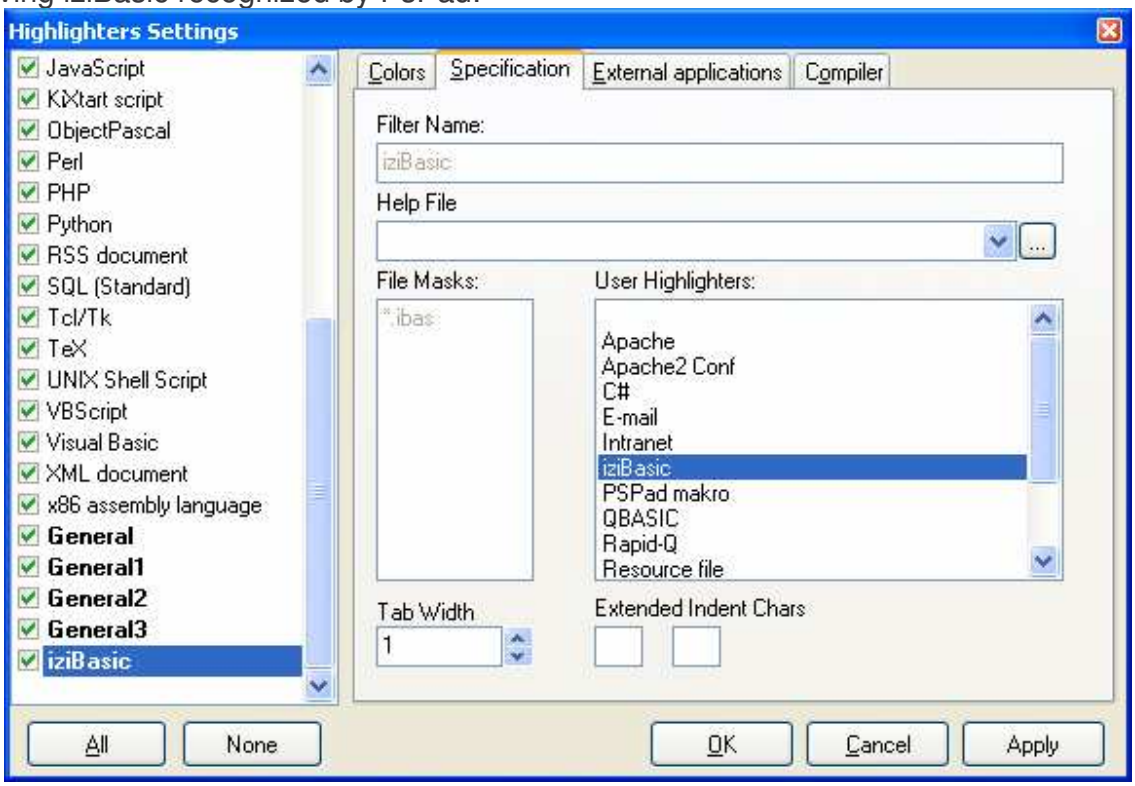
Step #1 – Installing the development environment

- ✓ First, download the PsPad self install executable and run it. By default, PsPad will be installed in the " *C:\Program Files\PsPad* " directory.
- ✓ Then, search for the " *C:\Program Files\PsPad\Syntax* " directory and copy the *iziBasic.INI* file in it.
- ✓ Download the *makedoc.exe* file and install it in any directory like, for example: " *C:\Program Files\MakeDoc* ". Copy the *iziBasic.BAT* file in this directory.
- ✓ Download the Palm OS Emulator (and adequate ROM) and/or the Palm OS Simulator, then unzip all files in a directory of your liking, for example: " *C:\Program Files\Palm Emulator* ".
- ✓ Eventually, create a new " *C:\Program Files\Palm Emulator\Autoload* " subdirectory (see below for its use).

Step #2 – Configuring PsPad

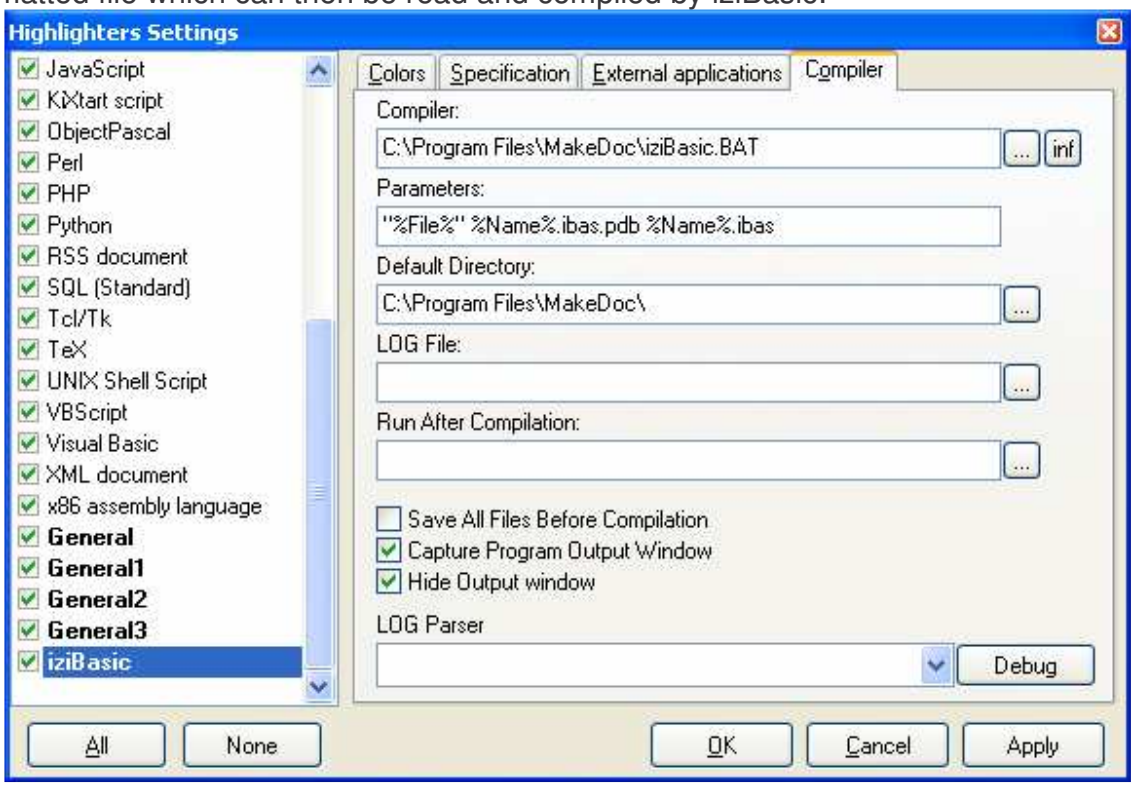
Launch PsPad and go to the Settings option in the menu bar, select Highlighters Settings. Please look at the screenshots below and set your environment as shown:

Having iziBasic recognized by PsPad:



Note the file Masks: "*.ibas". This means that all files with an .ibas extension in your PC will be recognized as iziBasic source code files. These files are just plain text files.

Getting PsPad able to “compile” is in fact to have it convert the source code to a DOC formatted file which can then be read and compiled by iziBasic:



PsPad will launch the *iziBasic.BAT* batch processing file and create the iziBasic source code DOC file from the currently edited source code in the following directory:
“ *C:\Program Files\MakeDoc* ”.

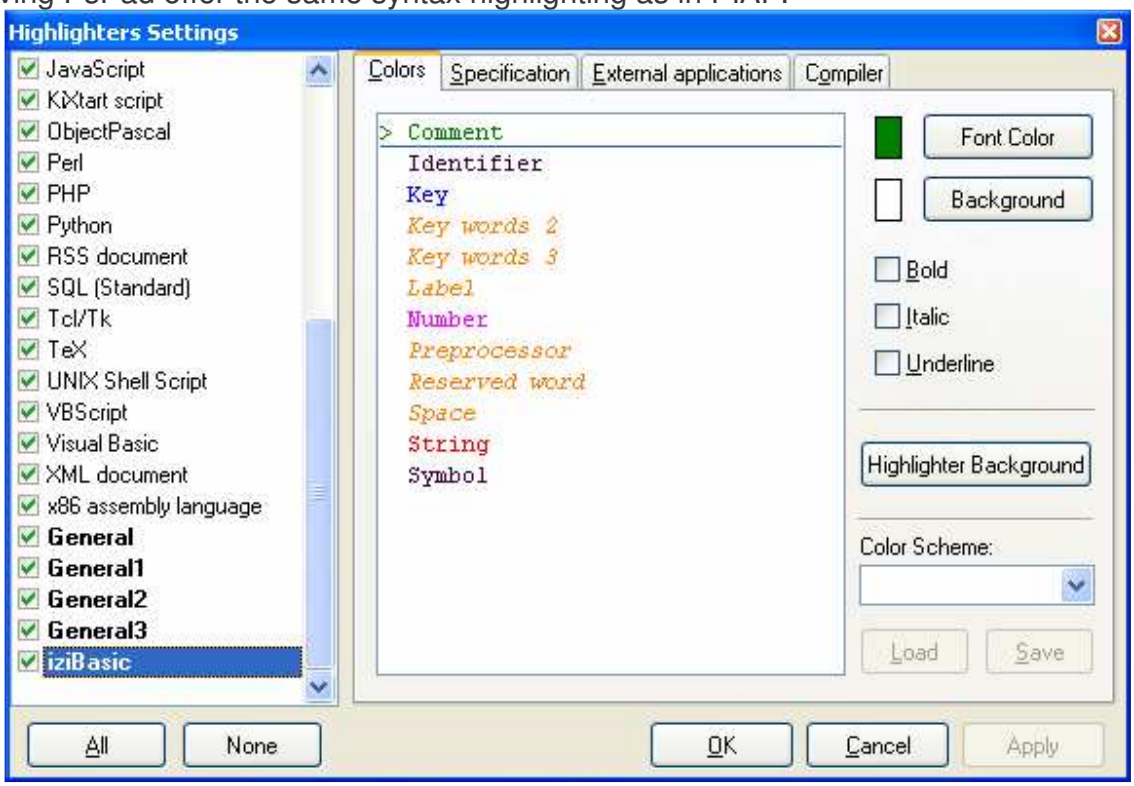
It may also chain with other actions, like shown in the *iziBasic.BAT* source code:

```
makedoc -n %1 %2 %3
REM copy %2 "C:\Program Files\Palm Emulator\Autoload"
REM "C:\Program Files\Palm Emulator\Emulator.exe" -load_apps %2
```

- ✓ If you wish to refresh an already opened Palm OS Emulator session with the new DOC file content, you will uncomment the second line (remove REM from it). Then, when choosing the reset option in the Emulator, each time the emulator will be reset, it will load back and refresh its memory with all files found in the “ *C:\Program Files\Palm Emulator\Autoload* ” directory.
- ✓ If you wish to launch a new Palm OS Emulator session with the current file loaded at launch time, you will uncomment the third line (remove REM from it).

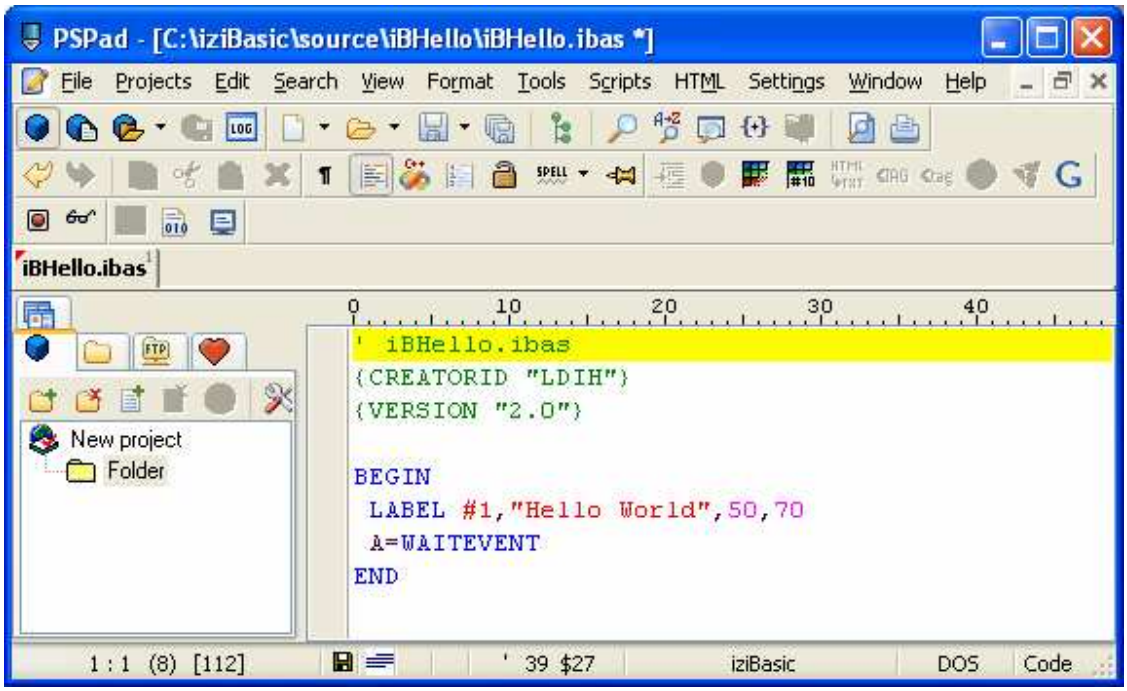
Note: using other command lines, you may just as well directly transfer the DOC source code to a real device by using one of the different available ways offered on the Palm platform.

Having PsPad offer the same syntax highlighting as in PIAF:



Note: I have identified on this screenshot in italic and orange all lines that are not used and do not need to be changed in the original settings you have.

At this stage, if you open a ".ibas" source code in PsPad, you should get this kind of result:



Note: all example files provided in the iziBasic package ZIP file are in the following format: *NameOfFile_ibas.TXT* so that they can be easily viewed by the default notepad application. They will have to be renamed to *NameOfFile.ibas* to be recognized as iziBasic source codes by PsPad.

You may then compare with the same source code shown in PIAF:

Apart from the "#1" color coding (which is considered as being a string by PsPad), everything looks exactly the same ☺

Step #3 – Exporting the source code to the Palm OS Emulator for compilation

Launching the DOC file building process and the export of this file to the Palm OS Emulator is now just a question of clicking on the button in the PsPad editor ☺

Step #4 – Working with the Palm OS Emulator

Whether you launch a new Palm OS Emulator session at each PsPad "compilation" process or just copy the edited file in the *"C:\Program Files\Palm Emulator\Autoload"* folder and reset the emulator to refresh its content, you will need to upload to the emulator the iziBasic environment to work and compile.

Therefore, you should put in the *"C:\Program Files\Palm Emulator\Autoload"* folder at least the iziBasic.prc application file. You may, at your convenience, add other application files like BIRD, PIAF, RsrcEdit, etc… and other required files for your development projects (resource files, other source code files like include files already ready for use, etc…).

Now that we have built a beautiful and powerful programming environment on a PC, don't you believe that coding and compiling directly on board of the Palm device is much more fun? ☺

Developing an iziBasic project on a Windows PC using the ConTEXT editor

Software & Files needed:

ConTEXT
The ConTEXT editor is also a great freeware programmer editor which works with almost any programming environments, providing highlighted syntax in code too. It is a light but still very functional alternative to the more sophisticated PsPad.
URL: http://www.context.cx/ Cost: free
Version available at the time I am writing this appendix: 0.98.3 - 9/6/2005

iziBasic.chl / iziBasic.BAT files
I provide these files which are the configuration files for ConTEXT to provide the iziBasic highlighted syntax and the automation of the DOC building.
iziBasic.chl is to be found in the iziBasic ZIP distribution file, under the " *Editors Highlighters\ConTEXT* " folder.
iziBasic.BAT is to be found in the iziBasic ZIP distribution file, under the " *Editors Highlighters* " folder.

MakeDoc
MakeDoc converts a plain text file to a DOC formatted file that can then be transferred to a Palm device and read by any DOC reader.
URL: http://nature.berkeley.edu/~alyons/palm_cybercafe.html Cost: free

Palm OS Emulator and/or Palm OS Simulator
The Palm OS Emulator emulates a Palm device, up to OS 4.x. It is an "emulator" because it needs a real copy of an OS ROM to run.
The Palm OS Simulator simulates a Palm device, for OS 5.x and OS 6.x. It is a "simulator" because the OS is compiled to run natively with a PC processor.
URL: http://www.palmos.com/dev/dl/dl_tools/ Cost: free

Note: if you choose other directories but those provided in the following instructions, you will have to customize all further explanations accordingly.

Step #1 – Installing the development environment

- ✓ First, download the ConTEXT self install executable and run it. By default, ConTEXT will be installed in the " *C:\Program Files\ConTEXT* " directory.
- ✓ Then, search for the " *C:\Program Files\ConTEXT\Highlighters* " directory and copy the *iziBasic.chl* file in it.
- ✓ Install MakeDoc as explained in the PsPad tutorial above. Copy the *iziBasic.BAT* file in this directory.
- ✓ Install the Palm OS Emulator and/or the Palm OS Simulator as explained in the PsPad tutorial above.

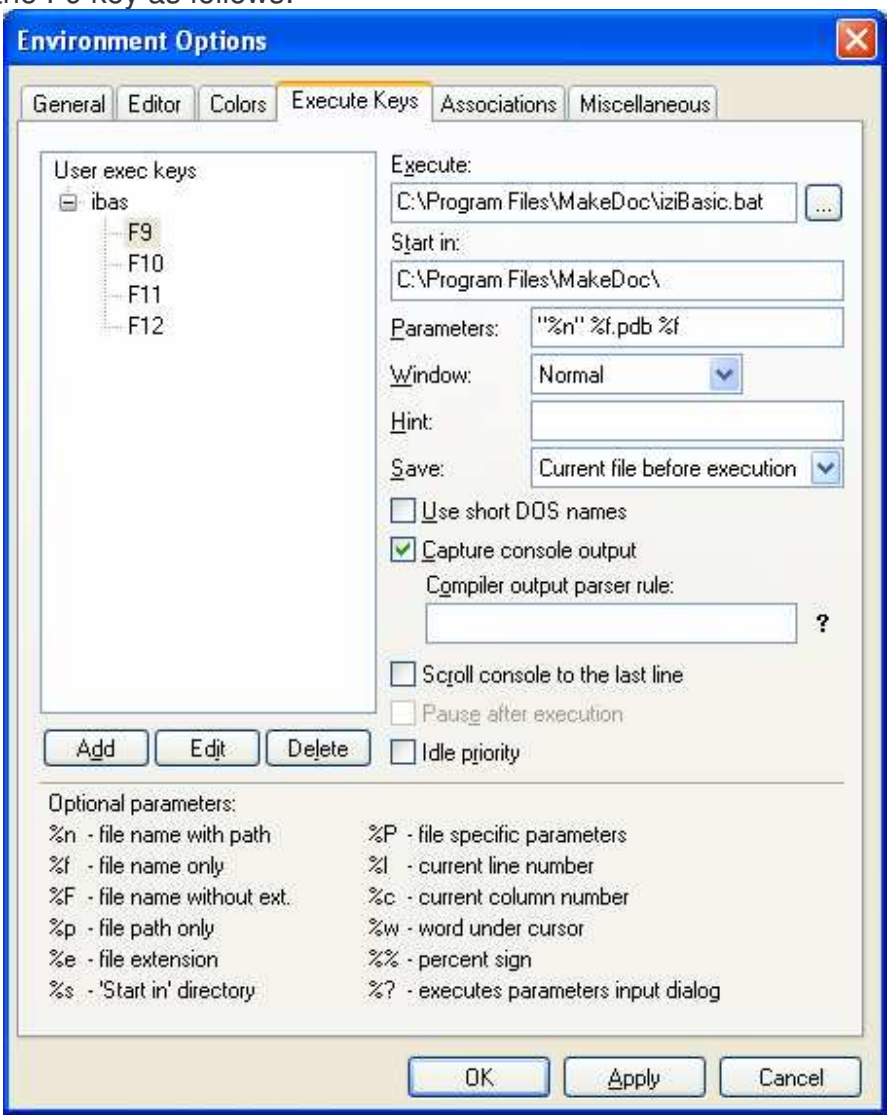
Step #2 – Configuring ConTEXT

On the difference with PsPad, ConTEXT is already fully configured to work with iziBasic as syntax highlighting is concerned.

Note the file Masks: "*.ibas". This means that all files with an .ibas extension in your PC will be recognized as iziBasic source code files. These files are just plain text files.

Now, getting ConTEXT able to "compile" is in fact to have it convert the source code to a DOC formatted file which can then be read and compiled by iziBasic and we will proceed in a very similar way to what we did earlier with PsPad, having ConTEXT launching the iziBasic.BAT batch processing file.

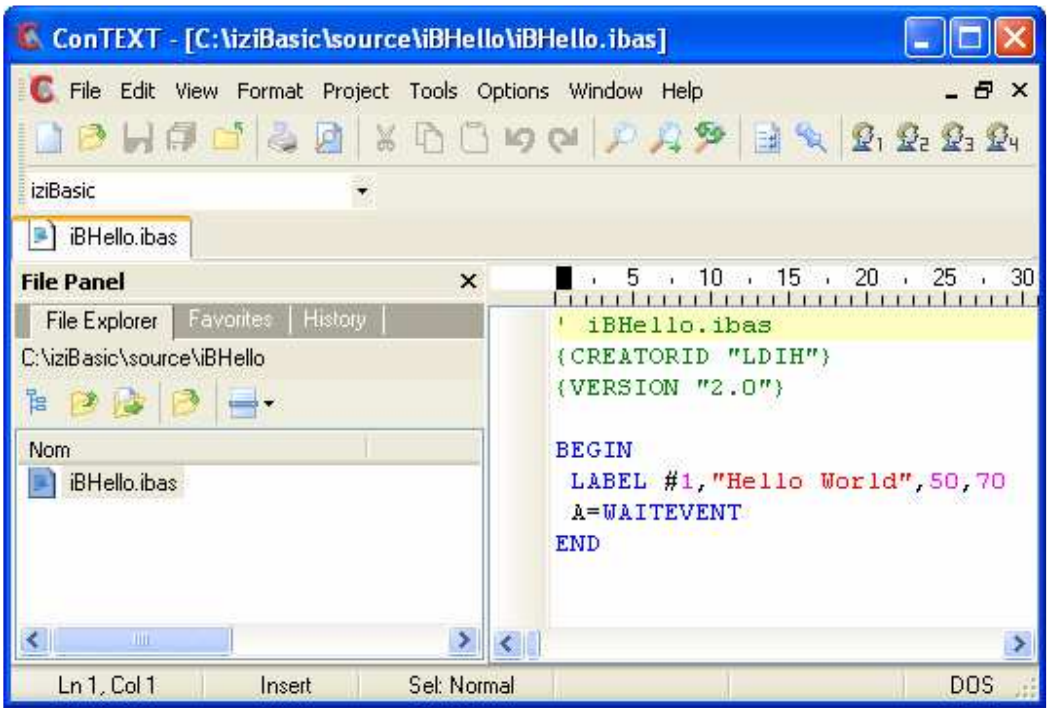
Go into the Menu, Options, Environment Options. Then, select the Execute Keys tab. Click the Add button, write “ibas” (without quotes) and then click the OK button. Then, fill in the F9 key as follows:



ConTEXT will launch the *iziBasic.BAT* batch processing file and create the iziBasic source code DOC file from the currently edited source code in the following directory: “ *C:\Program Files\MakeDoc* ”.

Note: all explanations about the iziBasic.BAT customization explained at this stage in the PsPad tutorial can be applied here as well.

At this stage, if you open a ".ibas" source code in ConTEXT, you should get to see this kind of result:



Note: all example files provided in the iziBasic package ZIP file are in the following format: *NameOfFile_ibas.TXT* so that they can be easily viewed by the default notepad application. They will have to be renamed to *NameOfFile.ibas* to be recognized as iziBasic source codes by ConTEXT.

You may then compare with the same source code shown in PIAF:

And yes, everything looks exactly the same ☺

Step #3 – Exporting the source code to the Palm OS Emulator for compilation

Launching the DOC file building process and the export of this file to the Palm OS Emulator is now just a question of clicking on the  button in the ConTEXT editor ☺

Step #4 – Working with the Palm OS Emulator

Again, the same explanations as those given for PsPad apply here.

Now that we have built another beautiful and powerful programming environment on a PC, don't you still believe that coding and compiling directly on board of the Palm device is much more fun? ☺

Editing an iziBasic project on a Windows/Linux/Mac computer

Please let me remind here that I do not know very well the Linux Operating System and even less the Mac one. As a consequence, the explanations here below will be quite limited.

Software & Files needed:

Sun Microsystems' Java Runtime Environment
Sun's Java Runtime Environment (JRE) is a cross-platforms virtual machine running Java scripts. iziEditor was developed on purpose using the Java language to ensure this cross-platforms compatibility.
URL: http://www.java.com/en/download/ http://www.pspad.com/en/
Cost: free
Version available at the time I am writing this appendix: 5.0

iziEditor
Khertan, who is a "heavy" iziBasic user, is working on developing an editor dedicated to iziBasic and PP (for PP applets). iziEditor is still in a very early development stage but it is already very functional, providing syntax color highlighting and load & save Text / DOC files capabilities.
URL (French): http://www.khertan.net/index.php/2005/08/16 Cost: free
Alternative URL (English): http://sourceforge.net/projects/izieditor/
Version available at the time I am writing this appendix: 0.1ß4

Palm OS Emulator
The Palm OS Emulator emulates a Palm device, up to OS 4.x. It is an "emulator" because it needs a real copy of an OS ROM to run.
For Windows users, please follow the instructions given in the above example using PsPad.
For Linux and Mac users, I have no idea of where to download and how to install the adequate version of this emulator. But a query in any online search engine should give you the answers you need to install and run it.

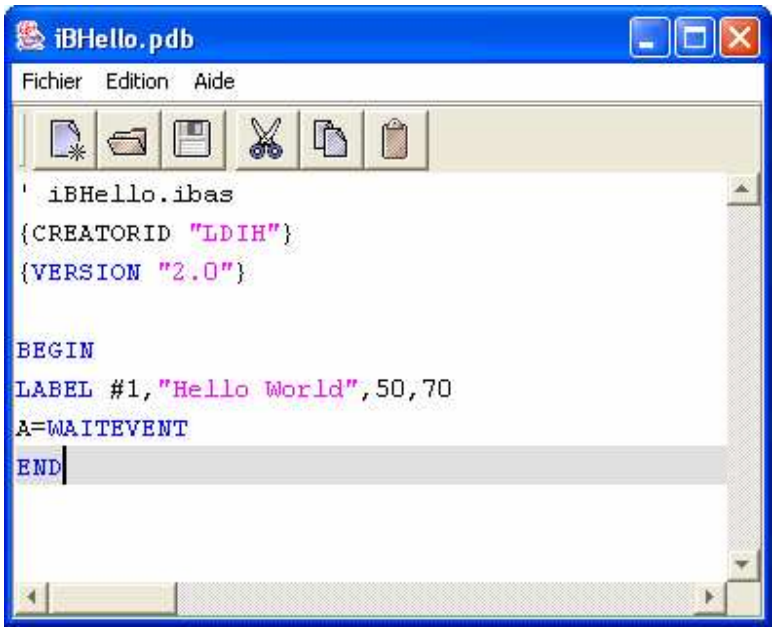
Step #1 – Installing the development environment

First, install Sun's JRE. Please follow their instructions as the installation procedure may change depending on the target Operating System.

iziEditor (iziEditor.jar) is just a single file (a compressed set of Java script files to be precise) that you may store anywhere in your hard disk and launch following your Operating System's procedures (for instance, a double click on this iziEditor.jar file is enough for Windows once JRE was installed).

Step #2 – Using iziEditor

iziEditor is used in a very similar way to any other editor. Let me just remind that it already offers syntax color highlighting and load & save Text / DOC files capabilities. These last load & save capabilities are especially convenient, when compared to the previous PsPad example, as it avoids the use of another tool like MakeDoc.



Appendix #11: iziBasic Versions history / log

Next versions roadmap

- Implement the CLIPBOARDADD statement
- Implement the slider GUI object
- Study the opportunity to open access to the 256 bytes buffer to share with global variables in the so called "PP applets"
- Study how to generate hidden applications, meaning applications with no form
- Study how to implement further connectivity (data IR beaming, Serial, Network…)
- Study how to implement built-in files access to external memory cards (SD Cards, Memory Sticks…)
- Implement a sophisticated Boolean parsing capability for compilation of what is currently "v|n TestOper v|n" and "c|t TestOper c|t" in conditional statements. At this stage, I'd rather study how to re-engineer the parsing capabilities of iziBasic to allow complex expressions in all areas (this would be a huge work, easy to perform but requesting to write back most of the compiler and the virtual machine code!)
- Implement any good idea that you would have ☺ … Current users will already find many of their ideas implemented in the previous releases!

v6.0 (11/04/05)
- Bug fix: iziBasic would crash when launched if the Memo Pad database had not been initialized by running the Memo Pad application at least once.
- Bug fix: in high resolution, the 20 text console lines (introduced in v4.0) would not always scroll correctly after an INPUT statement.
- Bug fix: the CLS console statement would not clear the screen in one case. Also optimized the code for the PRINT statement.
- Bug fix: the END statement, when included in a loop (IF, WHILE…), would cause an endless loop requiring a soft reset of the device.
- Bug fix: LTRIM$, TRIM$ and RTRIM$ could eventually get into an endless loop requiring a soft reset of the device if the string parameter was empty.
- Bug fix: objects are no more limited to the [1..255] range and can now be in the [1..999] range as expected and stated in the documentation.
- Bug fix: when the facultative target Creator ID was not specified, COPY would use iziBasic's Creator ID ("LDIB") for destination file instead of the source file's Creator ID.
- Bug fix: COPY would raise a fatal error leading to a soft reset of the device when trying to copy an empty database.
- Bug fix: the tracking of being or not in the main form was not always performed correctly in the virtual machine; that could lead to system crashes when working with custom forms (opened using the OPENFORM statement).
- Bug fix: DESTROYing a NUMFIELD, TEXTFIELD or TEXTFIELD$$ now frees the field's content in memory.
- Bug fix: a memory leak was generated (only a warning in the Palm OS Emulator when adequate debug option was checked).when quitting an application using menus, but with no side effect in real devices.
- Bug fix: calling SETFONT with either of the 128 or 129 values, or with the 130 or 131 values, was not behaving correctly.
- Enhanced integration of RESOURCEFILE by checking duplicate resources. Duplicate resources are no more added to the application and a warning is displayed.
- Enhanced the VAL function by having it consider leading spaces (" ") as zeros, so that VAL(" 10")=VAL("010")=10. Previously, a space character was considered as any other character but a digit so VAL(" 10") would have returned a zero value.
- Extended the INPUT statement from 23 to 63 characters.
- Optimization: if the personalized ABOUTBOX is not needed, its resource is no more integrated in the compiled application.
- Extended the scope of the CLOSEFORM and OPENFORM statements so that they can now handle the main form (which is, by default, built and displayed when the program is launched).
- Added a "Source Code Skeleton generator" wizard in the Options.
- Added another new option to allow pausing compilation when CHAINing source codes.
- Added a new POPUPCHOICE statement, which is very similar to the existing LISTCHOICE statement, to manage "real" popup lists (following Palm GUI guidelines).

(…continued on next page…)

- The list of items in a POPUCHOICE or a LISTCHOICE statement may now also be a pointer to the A$() array, starting at index A$(n) and stopping at the first empty index.
- Added a new UPDATECHOICE statement to allow changing the selection list of items in a POPUPCHOICE or a LISTCHOICE object.
- Just to mention it as this is a minor cosmetic enhancement, the drop-down list of iziBasic source codes that can be compiled is now resized to fit the number of source codes to display if there are less than 12 source codes available.
- Another minor cosmetic enhancement: the default about box now shows “Your Application is powered by iziBasic” instead of “Your Application was built with iziBasic”.
- Two and many more SELECT CASE / END SELECT statements can now be imbricated one into another.
- Implemented the SCROLLBAR object. Updated the existing GUI statements and functions impacted by this new object (DESTROY, HIDE, SHOW and UPDATEVALUE).
- Added the new TEXTFIELD$$ and GETFIELD$$ statements to manage bigger fields with the MegaString.
- Reading deleted records in a database no more generates a FILEERROR but returns an empty value, string or MegaString. This is more convenient to scan a database in the INPUT file access mode.
- Enhanced the MegaString’s INPUT$$ and PRINT$$ file reading and writing statements with an additional facultative parameter, to allow reading and writing sized blocks of data and no more only data ended with the CHR$(0) character.
- Added the new FIELDCOPY, FIELDCUT, FIELDPASTE and FIELDUNDO statements in the GUI module to provide fields management.
- Added a new GETFOCUS function in the GUI module.
- Implemented a first statement for connectivity, starting with IR (InfraRed) beaming of files. This statement was named BEAMFILE.
- Optimized the handling of fatal errors in the virtual machine.
- Added a new appendix (#9) to the user manual to show ways of working on an iziBasic source code on a Windows PC or on any Windows/Linux/Mac computer.
- Added the new ViziBasic editor to the Options editors list and set up the link for automatic compilation of source codes generated by ViziBasic (*ViziBasic is a new add-on project to iziBasic*).
- Upgraded the iBAddress sample program to use the new POPUPCHOICE statement.
- Removed a useless line in the iBClock source code.
- Bug fix: the small images checkbox now refreshes when setting to default values in the NekoCat sample program’s options.
- Added a new full sample program: tinyBasic. This is a small BASIC console interpreter written in iziBasic ☺ Also give a look to the new Memo2tinyBas GUI sample program developed to ease the keying and loading of tinyBasic source codes.
- Also added a simple sample source code, IBDim2, which is a simple application to show how to deal with 2 dimensions arrays in iziBasic, like Array(10,5), when iziBasic "only" offers access to the A(n) array (1 dimension).

v5.2 (05/24/05)
- Bug fix: the SQRT function returns back correct values (this was a regression since one or few iziBasic version(s) due to a change in the PP compiler and not ported to the Math library used to build iziBasic)
- “Very weird” bug fix: the compiled program could freeze or come to soft reset the device in some very particular cases, due to a bug in the final compiled code optimizer (well, it took me one month to figure this one out! This is why I qualify it as being “very weird”…)
- Bug fix: the SELECT CASE / END SELECT statements set would only work with the A-Z and A$-Z$ variables, not with the Numbers and Text variables with more than one character defined with the DIM statement
- Bug fix: UPDATEVALUE would not work on some GUI objects (PUSHUBUTTON and CHECKBOX) after an UPDATETEXT was made on these objects
- Bug fix: UPDATEFIELD would keep some heap memory space locked
- Bug fix: any operation on objects in a form opened by OPENFORM would cause a crash unless a FLUSHEVENTS or an event loop (DOEVENTS, WAITEVENT) was applied just after the OPENFORM statement
- Added a new {INCLUDE t} compiling statement, to include one source code into another one and have the iziBasic compiler parse the source codes accordingly
- Added support for global variables in the so called “PP applets”, for up to 256 bytes of global variables (256 bytes seems already large enough, but this could very easily be extended on request)
- Added a new sample program to demonstrate the use of the freeware CPDB library (http://cpdb.ppcompiler.org/) using a “PP applet”

v5.1 (04/04/05)
- Bug fix: compilation would crash in some cases when iziBasic was not installed using the standard HotSync procedure but using some third party tool instead (RescoExplorer for instance).
- Bug fix: in some cases, the compiler generated a misalignment between the A() array and the %var% variables (Numbers variables with more than one character) in the stack and, with no surprise, the same bug was found for the A$() array and the %var$ variables (Text variables with more than one character).
- Bug fix: now Numbers and Text variables with more than one character (%var% and %var$) can be passed among CHAINed programs and are also considered as global variables like the A-Z and A$-Z$ default variables.
- Bug fix: in the case a comment was added with a “ ‘ ” character in front of a “ : ” statements delimiter (so, the “ ’: ” sequence, the comment was ignored and compilation would give unpredictable results.
- Menu’s open and close events are now tracked. The MENUITEM function returns 0 if the menu was just opened or -1 if the menu was just closed.

(…continued on next page…)

- Extended the FLUSHEVENTS statement capabilities by adding a new facultative parameter to it.
- Upgraded the auto-repeating queuing of hard button keys events management loop in iziBasic's virtual machine.
- Added a final code optimization in the math parsing routine (when using the {PARSER ON} compiling directive) for a smaller application size and a faster execution at runtime.
- Added key shortcuts to launch different controls in iziBasic's interface.
- With SrcEdit, iziBasic can now receive from SrcEdit which source code to compile, start compiling automatically (without needing to press the [Build it] button) and send in return to SrcEdit the line where a compilation error was found (but not the error message because SrcEdit does not offer to manage this information) so that SrcEdit can open the source code directly on the line where the found error is.
- Added bookmarks in the user manual for a much easier browsing in the documentation.

v5.0 (02/23/05)
- Bug fix: if the console high resolution font was not found and the device had a high resolution screen, the vertical spacing of the console lines was not managed correctly.
- Bug fix: the EOF function was not always returning properly the end of file reach.
- Weird bug fix specific to Palm OS 5 causing a crash (prior versions of Palm OS do not complain!) in the handling of GUI objects.
- Bug fix in the Math parsing routine: calculations with AND, MOD, OR and XOR followed by an opening parenthesis were giving a syntax error when the expression to parse was semantically correct.
- Bug fix: in some cases, the iziBasic compiler was too permissive with the check of the statements separator character (:) and would allow statements not to be separated by this character (then things like the Pascal looking statement "WHILE test DO" were accepted).
- Split iziBasic's virtual machine (which was ~68 KB in version 4.2) into several independent modules so as to decrease the programs size. Apart from a core module (~36 KB), the other modules are now put in the application only if required.
- Changed the iziBasic's virtual machine from the 8 bits architecture (that had been built back at the beginning of the iziBasic project!) to a 16 bits one.
- Just to mention it, the 2 previous changes just above required a full revamping of almost the whole iziBasic structure and source code!
- The architecture change allowed me to extend the current data stacks to much wider sizes. The sizes are now limited by the dynamic RAM available in the devices and their Palm OS version when they were previously limited by the dynamic RAM available in a Palm OS 3.0 device. The sizing is managed by the MINOSVERSION compiling directive and the memory capacity of the developer's device.

(…continued on next page…)

- Upgraded the POKE statement as well as the FRE and PEEK functions to reflect the architecture upgrade.
- iziBasic now reserves the required space in the Numbers Stack for the PARSER ON compiling directive instead of using some space in the free stack on top of FRE(4). The PARSER memory usage in the Numbers stack is given for information in between brackets in the compilation report.
- Reengineered the IF statement to remove its previous limitation of accepting a GOTO only in the same THEN / ELSE, THEN / ENDIF or ELSE / ENDIF statements block. As a nice side effect, the IF statement is executed much faster in iziBasic's virtual machine. To get an idea of the speed improvement, the Bench2 benchmark (see sample programs source code) returns a ~30% decrease of execution time over this same program compiled with version 4.2 of iziBasic.
- The SELECT CASE / END SELECT, REPEAT / UNTIL, WHILE / WEND, FOR / NEXT instructions execution in iziBasic's virtual machine were also optimized.
- Implemented Numbers and Text variables with more than one character (so available in addition to A-Z and to A$-Z$) by extending the DIM usage to define these variables.
- Extended the labels and the new variables (see above) definition stack to 255 items (was 70 items before).
- To be homogeneous with the Basic language and the rest of iziBasic, labels are no more case sensitive, this means that _MyLabel and _MYLABEL are now the same.
- Added a final check for undefined labels in compiler (in the case a GOSUB or GOTO statement points to a label which is never defined afterwards).
- The ABOUTBOX and MENU statements can now be set up and changed dynamically at runtime.
- Upgraded the LISTCHOICE statement in the GUI module to work with index for initial selected item and the UPDATEVALUE to work with LISTCHOICE.
- Added the new OPENFORM, CLOSEFORM, SETFOCUS, FLUSHEVENTS, KEYBOARD, SAVESCREEN and RESTORESCREEN statements in the GUI module.
- Added a new SETRES statement in the Graphics module.
- Added a new SOUND statement in the Sound module.
- Added a new PARTIAL parameter to the KEYEVENTS compiling directive.
- Added a new CONSOLEFONT compiling directive for a potential 7 KB application size decrease.
- Extended the SECUREFILES compiling directive: when set to OFF, if a 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.
- Added a new appendix (#8) in this user manual to explain iziBasic's memory structure.
- Upgraded the iBAddress sample program to take benefit of the new SETFOCUS and KEYBOARD statements.
- Bug fix in the Matches sample program: the "no animation" option would not work since I added the crackling sound of burning matches in version 4.0

v4.2 (01/19/05)
- Bug fix: MENU would popup but not react to user input if activated by a tick on the title
- Bug fix: MENU navigation would not behave in the standard way when navigating with the up and down keys
- Enhanced syntax error tracking in some parts of the compiler
- Enhanced the GPRINT statement to now have it being able of drawing some characters with a given back color in addition to the specified text color
- Added a new PGET function in the Graphics module
- Added a new UPDATEVALUE statement in the GUI module, to allow updating the status of checkboxes and repeat buttons, and upgraded the source code of the NekoCat sample program with this new functionality
- One more time, I highly reengineered the source code of iziBasic which led to a new 2% decrease of the virtual machine size code since last version and to a faster execution of applications for events driven ones
- Apdi2003 upgraded again the French version of the user manual

v4.1 (01/14/05)
- Bug fix: the SELECT CASE v|c / CASE n|t / CASE ELSE / END SELECT group of statements could only be used once in a source code
- Bug fix: MENU would not work correctly
- Bug fix: in some devices equipped with Palm OS 5 but not all of them, pen moves were not tracked correctly (Palm OS 5 bug for penMoveEvent?)
- Added a new Tutorial - building and linking resources (Appendix #5) – in this user manual. Thanks to Mike Featherstone for writing it. It is great to receive such a user contribution.
- Also added High Resolution icons for iziBasic. And again, this is a very kind contribution from Mike Featherstone. Thank you Mike.
- Apdi2003 very kindly translated the user manual of version 4 to French

v4.0 (01/04/05)
- Bug fix: calling the ABOUTBOX would return a 0 value event in DOEVENTS, which was expected, and also in WAITEVENT, which was not expected! Now it returns a 1002 value event
- Bug fix: COPY would return a SysFatalAlert (and require a soft reset!) when copying a file with empty records to be deleted during next HotSync
- Numbers can now be passed in exponential format ([-]n.nnnnnnne[-]nn)
- Added a CALLPPARM$ function to give access to PP ARMlets from an iziBasic program. This opens wide possibilities of "*full speed*" routines!
- Implemented a so called "MegaString" which is 4 Kbytes big. This mainly allows reading & writing files records bigger than 63 characters. As a consequence, I added:
    - CLEAR$$, INPUT$$, PRINT$$, PUTCHAR$$ and PUTSTRING$$ statements
    - LEN$$, GETCHAR$$ and GETSTRING$$ functions
- Implemented new FINDFIRST$ and FINDNEXT$ files functions
- Upgraded the RUN statement with a new facultative parameter to pass to the target program, and implemented a RUN$ function to retrieve the passed parameter
- Added a new {SECUREFILES ON|OFF} compiling directive and upgraded the OPEN statement to take into account this new option
- Implemented {DEFINE t}, {IFDEF t}, {IFNDEF t} and {ENDIF} conditional compiling directives
- Made a technical upgrade to change iziBasic's resources from "tSTR" to "iBas" (this change has no influence for the iziBasic programmer)
- Added a new ADVICEBOX statement in the GUI module to display "tSTR" resources
- Upgraded the SETFONT statement to have it being able to work with custom fonts, implemented a new FONTWIDTH function
- In the GUI module, implemented a MENU statement and a MENUITEM function
- Implemented SELECT CASE v|c / CASE n|t / CASE ELSE / END SELECT statements in the Core module
- Implemented a new facultative parameter for the INPUT statement, which allows to capture directly a user input on the line of the requested input and which does not open the default input field and the wait button
- Changed the console font to a custom fixed font and extended the console display to 13 lines of text in low screen resolution and to 20 lines in high screen resolution (it was 11 lines previously no matter what the screen resolution was). Thanks to Philippe Guillot for allowing me to use in iziBasic the fixed fonts he designed for Piaf.
- Implemented a new GPRINT statement in the Graphics module
- Implemented a new PLAYWAVE statement in a new Sound module
- Massively reengineered the source code which led to a 10% decrease of the virtual machine size since last version, even with all new features of this new version!
- Added crackling sound of burning matches in the Matches sample program
- Upgraded the NekoCat sample program to work with high resolution images
- Upgraded a few console sample source codes to display nicely with the new fixed fonts
- Added a new GUI sample program with full source code: iBChristmas

(…continued on next page…)

- This is not a real iziBasic centric item: I worked on upgrading PIAF (PP's doc editor) to adapt it to editing iziBasic source codes (with color coding capabilities) and launching the iziBasic compiler. But, this was a piece of my work on the iziBasic project!
- With PIAF, iziBasic can now receive from PIAF which source code to compile, start compiling automatically (without needing to press the [Build it] button) and send in return to PIAF the line where a compilation error was found and the error message so that PIAF can open the source code directly on the given line and highlight the found error.
- Upon the request of many users, I did a "*work of slave*" (I am not sure of this translation from French!) on this user manual to provide descriptions and explanations for all statements and functions, and also to provide an alphabetical reference of all reserved keywords. As a consequence, the number of pages has doubled in this user manual… I hope that it will be useful!

v3.2 (12/15/04)
- Bug fix: ABOUTBOX would not work correctly with string variables (A$-Z$)
- Bug fix: STR$(v|n , v|n) would not return the correct string for values comprised in the]-1..0[interval
- Bug fix: compilation would hang and freeze if a stack overflow occurred
- Corrected a few minor typo mistakes in this user manual

v3.1 (12/02/04)
- Bug fix: in one weird case (that seemed to be random!), compilation would break on a syntax error with "garbage" characters
- Bug fix: the IMAGE statement can now work with the default images provided as shown in the IMAGEBUTTON statement
- Pen events are now trapped by the DOEVENTS and WAITEVENT functions
- Added a new {KEYEVENTS ON|OFF} compiling directive
- DOEVENTS and WAITEVENT may now track hard key presses if the KEYEVENTS compiling directive is set. They track for App Keys #1, #2, #3 and #4, HotSync Button on Cradle, Page Up and Page Down, Power On/Off
- Enhanced the overall events management runtime
- Changed the interpretation of the 2nd v|n in MID$(c|t , v|n , v|n) to be compliant with other Basic dialects: it is now the number of characters (or length) and no more the last character position; upgraded the source code of the following sample programs accordingly: ToDos, iBAddress, iBClock and Numerus
- Added 3 new defined constants: FALSE, TRUE and, for the fun of it, MAYBE ☺
- n (Number) can now be a Defined Constant
- Added an advice / trick in this user manual to prefix labels with a given character (underscore) or set of characters to avoid having labels starting with a reserved keyword; upgraded the source code of all sample programs accordingly
- Added a warning in this user manual about the use of GOTO within an IF statement
- Added a new GUI sample program with full source code: NekoCat

v3.0 (10/22/04)
- Bug fix: COS(0.8) was returning a Fatal Error in devices with Palm OS < 5 (and not in the emulator!), thanks to Ken for reporting it. This bug was due to a PP compiler bug which was fixed by the brilliant author of PP… so that I just needed to recompile iziBasic with the latest version of PP
- Added {PARSER ON|OFF} compiling directive and math parsing capabilities
- Added the following new boolean math operators: = <> > >= < <=
- Changed the POP and POP$ functions to one POP statement
- Added the HIDE, SHOW and TITLE statements in the GUI module, as well as a new NOTICEBOX function
- Added the CLIPBOARDGET and CLIPBOARDPUT statements, as well as the BATTERYINFO and HOTSYNCINFO$ functions in a new System module
- Moved the GETOSVER$ function to the new System module
- Added the INSTRING, CHAR$ and WORD$ functions in the main functions module
- Added the RSORT A, RSORT A$, SORT A and SORT A$ statements, as well as the MEAN A, MIN A, MAX A and SUM A functions in the Arrays module
- Added the EXP constant
- Added the DO / LOOP statements, which have the same behavior as REPEAT / UNTIL as some users from other Basic dialects preferred this wording
- As expected from v2.0 (see below, italic text in v2.0 history log – Optimization paragraph), I optimized the virtual machine and the compiler for all the functions, for a smaller virtual machine footprint, smaller executable codes and a faster execution of programs
- Checked compatibility with Palm OS 6: iziBasic is Palm OS 6 compliant ☺
- Added a Tutorial appendix - my first iziBasic program in this user manual file, to help first time users get started in writing an iziBasic program
- Made various upgrades in some sample programs: iBClock, Matches (was formerly iBMatches) and Numerus (was formerly iBNumerus)
- Added 2 new sample programs which read and display the Memo titles and the To Do list (with priority, status, due date) of the inbuilt Memo Pad and To Do applications

v2.0 (09/04/04)
- Bug fix: ABOUTBOX now accepts correctly from 1 to 3 string parameters, when it was only working with 3 parameters
- Bug fix: in some weird cases, iziBasic was crashing when being launched
- Enhancement: the database backup flag is now set by files creation statements, so that the databases created by an iziBasic program will be backed up to the desktop computer
- Enhancement: the backup flag was also set for the programs compiled by iziBasic so that your wonderful creations will be sent during the next hotsync to your computer
- Enhancement: the PENX and PENY values are also updated when the pen first touches the digitiser and when it is moved on the screen (previously only when the pen was lifted from the digitizer), also added a PENDOWN function to track the pen status
- Optimization:
    - of iziBasic's compiling engine which builds smaller executable codes
        the generated p-code is ~30% smaller in average
    - and of iziBasic's virtual machine which runs faster
        the Bench1 benchmark returns a 4.7% speed improvement
        the Bench2 benchmark returns a 3.4% speed improvement

    Note: more optimizations can be expected in a future release as I found other areas of improvement
- Added a few new math functions: ACOS, ASIN, DEGREE, LOG, POWER, RADIAN
    Note: other mathematical functions could be made available (like CosH, ArcSinH, DMS2Deg, Rad2DMS, etc…). Please feel free to ask for them ☺
- Added CALLPP$ function, this is a PP Code Segment call feature which opens all possibilities of the Palm development by direct access to the Palm OS APIs (in other words, you can now include "PP applets" in an iziBasic program)
- Added COLOR(v|n) function to capture forecolor and backcolor
- Added WAITEVENT function
- Added GRAFFITISHIFT, PUSHBUTTON and TEXTSELECTOR objects
- Added UPDATEFIELD, UPDATELABEL, UPDATEPOS and UPDATETEXT statements
- Interface improvements:
    - iziBasic now remembers the last compiled source code name, so as to ease the build and try procedure
    - if compilation succeeded, iziBasic now offers to run the just built program
    - added a button to launch your favourite editor (Memo Pad or one of these DOC editors: PIAF, QED, SiEd, SrcEdit)
- Updated and upgraded this user manual file in many areas
- Made various upgrades in the sample programs and added a new iBHelloPP demo source code

v1.0 (07/22/04)
- initial release for Palm OS