Elena Kolchina's

Smart Basic

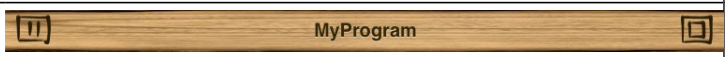
Reference Manual

version 5.8

collected and edited by 'Dutchman' Ton Nillesen, 20190219

Bars&Buttons

- ***Bars***

Start-page	
In Folder	
In Dropbox ready to paste	
In Editor	
Runtime	
Finished	

- ***Buttons in Top bars***

 New file	 Go to parent Folder	 Go to selected Folder
 Show Folder	 Search in text	 Go to Help-Info
 Start the program	 Open Debug window	 Stop the program
 Enter Editor		

- ***Buttons in Bottom bar***

 Delete file or folder	 Paste file or folder	 Cut to replace
 Copy file or folder	 Rename file or folder	 Make new folder
 Go to Dropbox	 Go to local folder	

Smart Basic

Reference Manual

version 5.8

collected from the internal documentation of the app
rearranged and edited by 'Dutchman' Ton Nillesen

modification date 20190219

Latest version of this manual is available at
https://www.dropbox.com/sh/zpsvd55g1iyjldj/AACDu5swXvXVRq26kigwAdqGa
or *http://bit.ly/1vwIPHn*

Document generated in free software office package: LibreOffice 5.4

Contents

Smart BASIC version 5.8 developed by Mr. Kibernetik
Welcome to support forum *http://kibernetik.pro/forum*
to share programs and discuss programming topics!

Personal notes

- ***File names***

Files, i.e. programs, created in the editor will get the extension '.txt' by default.

Apple however does not allow the up- and download facility in the app to download files with extension '.txt' from Dropbox.
Smart Basic, on the other hand, allows to edit and run program-files with other (although not all) extensions. E.g. the extension '.sb', initiated by Microsoft for 'Small Basic', can be used instead.

Other text-files for e.g. data or info can be named without an extension, only ending with a dot, e.g. "Read me." If the filename ends with a dot only, then it can be downloaded from Dropbox and be opened in the editor. Extensions with length≠3 however, as '.tx' or '.data' could also be used.

In order to separate program-files from data-files in the listing it is recommended to start **program-filenames with a capital and data-filenames in lowercase**. In this way the program-files will appear at the top of the listing.
Start the filename of a program under development with a hyphen '-'. It will then appear at the very top.

- ***Screen sizes***

Results obtained from PRINT SCREEN_WIDTH();"x";SCREEN_HEIGHT()

Device	Portrait	Landscape
iPhone 3.5-inch	320x436	480x276
iPhone 4-inch	320x524	568x276
iPad	768x980	1024x724

The height of the top-bar in Smart BASIC is 44 points.

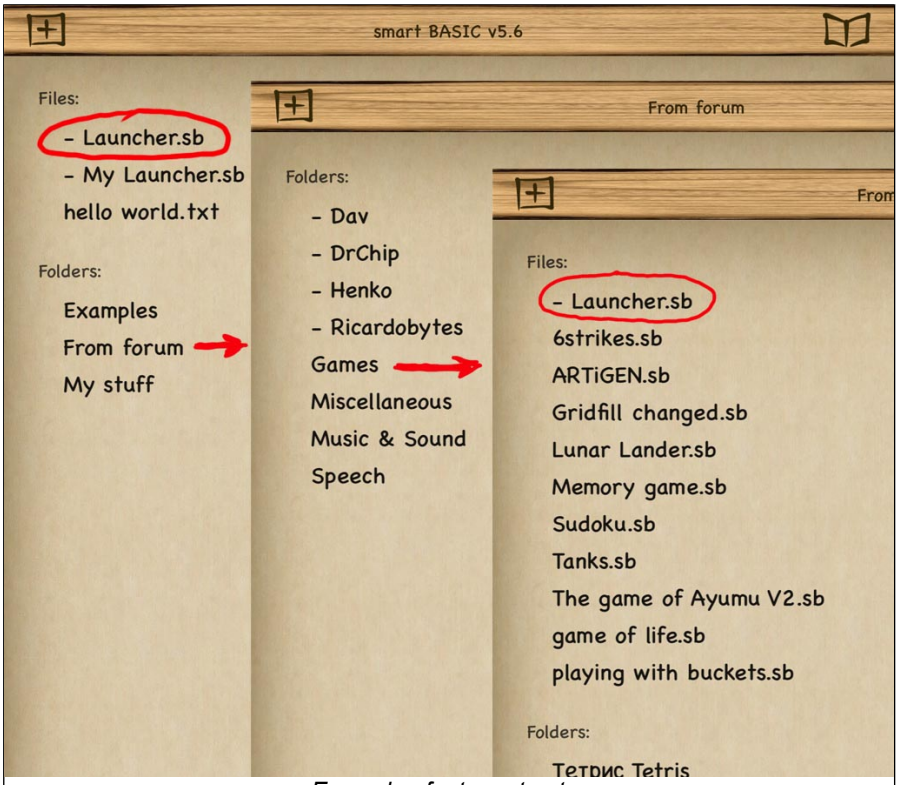
- ***Number accuracy***

Smart BASIC uses standard 64-bit floating point numbers in 1-11-52 bit patterns for sign-power-mantissa parts.
Any integer with an absolute value 2^53 or smaller will be represented exactly.
Smart BASIC operates floating numbers of double precision

- ***Tree structure***

Smart BASIC is designed to work with Dropbox cloud storage. The root directory however can not be transferred as a single folder. If you want to save your contents of Smart Basic to Dropbox, then you have to transfer the contents of the root piece by piece. As a consequence the contents of the root should be kept at minimum. After installation the app will contain two items in the root directory, the example program 'hello world.txt' and the folder 'Examples' which contains a wealth of example-programs. It can be restored to this default location if you by accident have made a fatal mistake during training wit these example-programs. Furthermore it is

wise to separate your own 'stuff' from the contributions of forum-members.
An example of such a tree is given in the following figure.



Example of a tree structure

- ***Launcher***

In the root-folder you'll see file “- Launcher.sb”. The arrows in the figure point to an encircled copy in the folder “/From forum/Games”. It is used to start the launcher, which makes program-selection and activating much simpler and faster.
The launcher is based on an idea from forum-member 'Ricardobytes' (*http://kibernetik.pro/forum/viewtopic.php?f=20&t=1499*) and contains the following code.

```
CALL ReturnToLauncher
'----- Functions
DEF ReturnToLauncher
path$="/My stuff/Launcher/Launcher.sb"
IF FILE_EXISTS(path$) THEN RUN path$
TEXT ! TEXT CLEAR
PRINT "Returnpath to launcher not found"
END DEF
'
DEF Launch$ 'function to read launcher-data from clipboard
```

```
'--- returns Id$
' Id$ is "desktop" or "basic" or "launcher"
' Line$ is Launcher DATA-text via Clipboard
Line$=""
IF Id$="" THEN
  WHILE CLIPBOARD_COUNT()<>0 AND txt$<>"launcher"
    CLIPBOARD READ txt$
  END WHILE
  IF txt$="launcher" THEN
    Id$=txt$
    IF CLIPBOARD_COUNT()<>0 THEN CLIPBOARD READ Line$
  ELSE
    Id$=Launcher$()
  ENDIF
ENDIF
RETURN Id$
END DEF
```

It will start the 'launcher' (=path$) which gives fast access to your favorite programs. It is intended to be included in a program which is started by the launcher and should recall the launcher if it is finished.
Some programs however are difficult to adapt for automatic return to the launcher. In those cases a copy of "- Launcher.sb" in the appropriate directory can be used to restart the launcher. That is illustrated in the figure by the arrows.

- ***Program structure***

The program structure should be such that the flow always ends at the 'END' command. The launcher can then simply be added just before the 'END' statement.
In the following dummy 'program' the main-loop ends via setting '.Quit=1' somewhere in the DO-loop. Another possibility is via flow-transfer to the exit-label: 'GOTO Finish' .
The bold line is essential for usage of the launcher.

```
'Dummy code with launcher
GOSUB Initialise
DO
  ……
  IF … THEN Finish
  …
UNTIL Quit
Finish:
IF Launch$="launcher" THEN {/- Launcher.sb}
END
```

The Launcher-folder on Dropbox contains several examples and information on usage etc. See *http://bit.ly/2bl7Hzl*

> If you **start the program in Basic**, **then it will return in Basic**.
> If you **start the program in Launcher**, either via desktop or in Basic,
> **then the program will return to Launcher**

See also coupling of launcher to desktop-icon in 'Notes' on page 76

Modifications since previous manual version 5.6

Version 5.8 is an update according to Dropbox requirements

- ***Additions and changes in this manual***
  - example on musical quarter notes and chords on page 62
  - Note on touch control in text mode on page 18
  - Note on string as return-value of user-defined function on page 15
  - Added subchapter "Sprite order rules" on page 55

1. Basics

An introduction to programming in Basic can be found on the forum:
http://kibernetik.pro/forum/viewforum.php?f=30

1.1. Variables

There are two types of variables in smart BASIC: numeric and string.
String variables have "$" sign at the end and can contain alphabetic and numeric characters.
Numeric variables can contain **real and complex** numbers.
Numeric variables with real numbers:

```
A = 10
B = 0.5E2
```

Numeric variables with complex numbers:

```
C = 2 - 3i
D = 1i
```

The floating point numbers are of double precision.
Numeric variable precision corresponds to 64-bit representation of 1-11-52 for sign-power-mantissa. Any integer with an absolute value smaller than 2^53 will be represented exactly.

String variable:

```
T$ = "Name"
```

String value can use quote symbol " by specifying double quotes "". Example:

```
N$ = """This is a quoted text"""
```

Command **LET** used in standard BASIC can be omitted, so:

```
LET X = 1
```

is similar to:

```
X = 1
```

1.2. Arrays

Arrays can be numeric or string with one, two or three dimensions.
If the command-description does not state what kind of array should be used, then any kind of array is possible. If array type is restricted then it is stated as "numeric array" or "string array".
Array should be declared before first use if its size is larger than 10.
Only real component of complex number is used in indices of arrays.

OPTION BASE N
makes array-indices start with [n]. Valid values are 0 or 1. Default is 0.
As strings are handled as one-dimensional arrays, this command has also effect on the index of the separate characters in the string.

OPTION_BASE ( )
returns current value, set by OPTION BASE command.

DIM ident(size)
Declares an array.
'ident' can be a numerical or string-identifier. 'size' determines the size of the array.
Several arrays can be declared on a single line separated by "," character.

```
DIM T$(20), B(50)
```

Size can be determined by a variable.

```
NumPoints=10
DIM PointX(NumPoints), PointY(NumPoints)
```

```
...
FOR i=1 to NumPoints-1
  PointX(i)=PointY(i+1)
  PointY(i)=PointY(i+1)
NEXT i
```

Example of a 2-dimensional array:

```
Lines=10 ! Columns=3
DIM Table$(Lines,Columns)
Table$(1,1)="Item")
Table$(1,2)="Origin"
Table$(1,3)="Value")
RESTORE TO TableContent ' Set pointer to list of data
FOR i=2 TO Lines
  FOR j=1 TO Columns
    READ Table$(i,j)
  NEXT j
NEXT i
```

GET DIM N XSIZE X YSIZE Y ZSIZE Z

gets sizes of "x", "y" and "z" dimensions of array [n] and saves them to numeric variables [x], [y] and [z].

You may specify only those dimensions which you need, for example:

```
GET DIM n XSIZE x
```

Sizes of each dimension in a multi-dimension array can be changed with just another DIM command. The number of dimensions cannot be changed in re-declaring. Speed of DIM command is equal whether it is first declaration of array or it is a re-declaration. Definitely it is faster to re-declare an array than clearing it with zeroes in a FOR/NEXT loop.

OPTION SORT ASCENDING

OPTION SORT DESCENDING

OPTION SORT INSENSITIVE

OPTION SORT SENSITIVE

Sets sorting of command SORT to be in ascending, descending, case insensitive or case sensitive order.

SORT M

Sorts numeric or string one-dimensional array [m]. By default sorting is performed in ascending order, case sensitive.

SORT M AS N

Sorts numeric or string one-dimensional array [m] and outputs sorting result as one-dimensional numeric array of indices [n], which indicate elements in array [m] in a sorted order. OPTION BASE command affects starting value in array [n].

SORT M TO N

Sorts numeric or string one-dimensional array [m] and saves sorting result to array [n]. Types of arrays [m] and [n] must be identical.

1.3. Expressions

Numeric math operations are: +, -, *, /, %, ^.

Operation % is a remainder of division, ^ is a power operation.

```
A = (2 ^ 3 + 1) * 2
```

Strings concatenation operation is: &

```
T$ = "Result is " & M$
```

Numeric to string and back conversion is automatic:

```
A = B$ + 5
B$ = A
```

Compact notation in assignments

```
A = A + B
```

can be written shorter:

```
A += B
```

for any numeric math operation or for string concatenation operation.
Thus, the following code:

```
C *= D + E
A$ &= "Text"
```

is equivalent to:

```
C = C * (D + E)
A$ = A$ & "Text"
```

Complex numbers

Math operations can accept real and complex arguments. smart BASIC easily understands complex and real numbers and properly calculates them.
Complex numbers are written as real part plus (minus) imaginary part with letter 'i' at the end:

```
A = 2 + 3i
```

Without imaginary part it is a usual real number:

```
A = 2
```

If real part of complex number is zero then you can write only imaginary part, for example:

```
A = 3i
```

In smart BASIC it is fine to write:

```
A = B + 2 – 4 * 3i + 10 – 2i ^ 2
```

1.4. Loops & Jumps

Only real component of complex number is used in cycles.

- ***Loops***

FOR / NEXT

```
FOR K = 0 TO 10 STEP 2
...
NEXT K
```

If parameter STEP is omitted then it is assumed to be 1.

WHILE / END WHILE

```
WHILE K < 10
…
END WHILE
```

DO / UNTIL

```
DO
...
UNTIL K < 10
```

- ***Jumps***

All labels are local, this means that every user-defined function has its own labels.

BREAK

interrupts WHILE/WILE END, DO/UNTIL and closest FOR/NEXT loops:

```
DO
...
IF K = 5 THEN BREAK
...
UNTIL K < 10
```

BREAK [x]

interrupts FOR/NEXT loop for variable [x]:

```
FOR K = 0 TO 10
...
  IF K = 5 THEN BREAK K
...
NEXT K
```

CONTINUE

jumps to next iteration of WHILE/WILE END, DO/UNTIL and closest FOR/NEXT loop:

```
DO
...
IF K = 5 THEN CONTINUE
...
UNTIL K < 10
```

CONTINUE [x]

loops to next iteration of FOR/NEXT loop for variable [x]:

```
FOR K = 0 TO 10
...
IF K = 5 THEN CONTINUE K
...
NEXT K
```

GOTO label

```
10 Y = X / 2
GOTO 10
Add: A = B + 1
GOTO Add
```

ON / GOTO

```
ON x GOTO 10, 20, 30
```

if [x] = 1 then goes to 10, if [x] = 2 then goes to 20 and so on.

ON / GOSUB

```
ON x GOSUB 10, 20, 30
```

Similar to ON GOTO, but calls subroutines instead of jumps.

1.5. Subroutines

GOSUB
Goto subroutine

RETURN
Return from subroutine
Examples:

```
Pi=3.141592653
A=Pi/2
GOSUB Hypotenuse
PRINT Result
END
Hypotenuse:
Result= SQR(SIN(A)^2 + COS(A)^2)
RETURN
```

Examples of calling subroutine at line number or label:

```
GOSUB 100
GOSUB MY_SUB
```

Examples of conditional subroutine selection by line number or label:

```
ON Result GOSUB 100, 130, 180
ON SIGN(Result)+2 GOSUB Increase, Compare, Decrease
```

1.6. Remarks and comments

There are three ways to make remarks or comments in your program.
Single line remarks, by using command **REM or** by using symbol ', for example:

```
REM comment
A = B + 1 REM comment
'comment
A = B + 1 'comment
```

You can comment blocks of text by surrounding it with symbols /* and */
for example:

```
/* comment begin
comment end */
```

1.7. User functions

Single-line definition:

```
DEF F (X,Y) = SQR (X^2 + Y^2)
```

Multiple-line definition:

```
DEF F (X,Y)
F = SQR (X^2 + Y^2)
END DEF
```

Example of arrays passed to functions as parameters:

```
DIM M (100), N(20,20)
DEF FUNC1 (X())
...
END DEF
DEF FUNC2 (Y(,))
...
END DEF
CALL FUNC1 (M)
CALL FUNC2 (N)
```

Function execution can be terminated by command RETURN.
Also RETURN command can return function result.

If the return-value is a string, then the function-name should end with '$'.
Example:

```
DEF spaces$(n) ' return string with n spaces
a$=""
FOR i=1 TO n
  a$&=" "
NEXT i
RETURN a$
END DEF
```

User function without parameters can be defined and called with or without "()".
All variables defined inside function are local and static.
Arrays passed to function as parameters are **references**, this means that this is the same array in function and in main code. So modifying the array has global effect.
NOTE: **Array cannot be re-initialized** inside the function when it is passed as a parameter to this function. So only **the contents can be changed, not the size**.
Other parameters are passed to functions by **value**.
In the following example the value of 'b' will not change.

```
b=10
PRINT b
CALL func(b)
PRINT b
END
REM --- User function
DEF func(a)
 PRINT a
 a-=1
 PRINT a
END DEF
```

Functions may be used recursively, i.e. the function can be called within the definition of the function. Example:

```
DEF factorial(n)
IF n>1 THEN factorial=n*factorial(n-1) ELSE factorial=1
END DEF
```

User-defined functions can be defined anywhere in the program, except within functions. Subroutines however can be used within functions and will be handled as local code.

1.8. Comparison and Logical operators

Comparison operators are: **=, <, >, <=, >=, <>**
Logical operators are: **AND, OR, NOT**
Comparisons are available for numbers and for strings.
For complex numbers available comparison operators are: **=, <>.**

IF / ELSE
Single-line:

```
IF A = 0 THEN GOTO 10 ELSE GOTO 20
```

Short version:

```
IF A = 0 THEN 10 ELSE 20
```

IF / ELSE / END IF
Multiple-line:
```
IF X < 0 OR Y > 0 THEN
  A = 0
ELSE
  A = 1
END IF
```

1.9. Scope variables

Although all variables are local, smart BASIC allows you to use any variable from any function inside any other function. For this purpose you can use scope variable syntax as "scope.name", where "scope" is variable definition scope and "name" is variable name. For example, if in your program F function is defined:
```
DEF F(X)
X2 = X^2
X3 = X^3
F = X2 + X3
END DEF
```
then you can get access to its local variables X2 and X3 outside of this function like this:
```
X = F(2)
PRINT X; F.X2; F.X3
```
Scope variable syntax allows you directly set function parameters and get any number of function results.
So called "global" variables, when you get access to variables outside the function, in scope variables syntax looks similar but without any scope:
```
DEF F
F = .X^2
END DEF
...
X = 2
PRINT F
```

- ***Notes from the forum:***

Grouping variables to virtual scopes

An attractive implementation of scope variables is grouping of variables to virtual scopes. It is not necessary that scope names can be only DEF function names. Your own scopes are created automatically when you declare scope variable.
For example, you can write:
```
SCR.WIDTH = SCREEN_WIDTH()
SCR.HEIGHT = SCREEN_HEIGHT()
```
This way you automatically create "SCR" scope. When you pause your program, all variables are listed grouped by their scopes, so you will see:
```
SCR:
WIDTH = …
HEIGHT = …
```
This approach can help you organise variables into scopes for better program layout.

Grouping arrays into records

Often arrays contain a lot of data with different identities assigned to certain indices. During programming it becomes then difficult to remember which index is assigned to a certain identity. To improve the readability, explicit identity-naming can be ap-

plied using arrays as scope variables arranged in a 'record' via a function definition. An example is given in the following program which operates on the 'record' “User”:

```
' named indices
' ---- Presets
Users=4
OPTION BASE 1
' ---- initiate content
CALL User(Users) ' initiate arrays
FOR n=1 TO Users
  READ User.Name$(n)
  READ User.Age(n)
NEXT n
' ---- display content
TEXT CLEAR
PRINT "**** Users ****"
PRINT "Name","Age"
PRINT "————————————"
FOR n=1 TO Users
  PRINT User.Name$(n),User.Age(n)
NEXT n
END
' ===== Functions and Data =====
DEF User(n) ' organised as 'record'
  DIM Name$(n), Age(n)
END DEF
DATA "John",42,"Mary",24,"Iwan",33,"Katinka",22
```

2. Input & output

Audio-output is handled in a separate chapter: 9Music, sound and speech on page 59.

2.1. Touch handling

Smart BASIC supports 11 simultaneous touches. Each touch is tracked individually and does not depend on other touches, this means that one touch can be active while other touches can appear and disappear at the same time. To distinguish touches from each other they receive numbers from 0 to 10. Touch receives its number as soon as it appears and keeps this number while it is active. Assigned number is minimal from currently available numbers. There is command GET TOUCH and functions TOUCH_X() and TOUCH_Y() to get touch coordinates. If touch with specified number does not exist then returned coordinate value is -1. If touch with specified number is currently active then its actual coordinates are returned.

Note from the forum:

> *You use x = Touch_x(i) in which 'i' is the number of touch.*
> *Total there can be 11 simultaneous touches, with touch numbers from 0 to 10. So, command x = Touch_x(0) means that you are asking x coordinate of touch number 0. If you get x = -1 then this means that there is no touch with this number.*
> *What about touch numbers.*
> *First touch gets number 0, so Touch_x(0) will give you proper x-coordinate of first touch. If you keep touching and then second finger also touches the screen (so now you get two simultaneous touches), then this new touch gets*

number 1 and Touch_x(1) will give you x-coordinate of second touch. If you release your first touch (touch number 0) then Touch_x(0) will return -1, but Touch_x(1) will continue to give you valid x-coordinate of your second touch (touch number 1) because it is still active. Any new touch gets smallest available number, so if you touch again then the new touch will be number 0.

Normally touches work only in graphics view.
Note from Rbytes on the forum:

The documentation for TOUCH commands says that they don't work in text mode. But they do under special conditions, opening up new possibilities for user control on a TEXT screen.
The trick is to create a field on the TEXT screen. It can be as tiny as 1 x 1 points and located at a screen corner or entirely outside the visible screen area (ie. made the field x coordinate a negative number). That activates the whole screen for detecting touches!

GET TOUCH N AS X,Y
gets "x" and "y" coordinates of screen touch number [n] to variables [x] and [y]. If there is no active touch with this number, [x] and [y] variables get value -1. First touch gets number 0.

TOUCH_X (N)
returns "x" coordinate of screen touch number [n]. If there is no active touch with this number, it returns -1.

TOUCH_Y (N)
returns "y" coordinate of screen touch number [n]. If there is no active touch with this number, it returns -1.

2.2. Input from keyboard or file

• Input from keyboard

INKEY$ ()
returns output from physical keyboard. Physical keyboard support should be turned on (OPTION KEYBOARD command) for this function to operate.

INPUT variable, ..., variable
performs input of values for specified variables in a special input field at top of text screen.

```
INPUT A, B$
```

You can specify additional inquiry before variable using ":" character:

```
INPUT "What is X?":X
```

User interface with INPUT command

In 'classic' implementations of BASIC the user types his response on INPUT directly after the prompt. That type of input-request is still common practice in 'terminal' mode for iOS and DOS and is also used in other recent Basic implementations.
In Smart Basic however the response on the INPUT command is required in a separate field at the top of the screen, which is rather clumsy for both the user and the programmer.

- The attention of the user has to be attracted to the top of the screen.

Furthermore:

- The background of the input field does not change with the screen color.

- The prompt is less visible due to low contrast.
- The cursor is positioned before the prompt which is strange but acceptable.
- The prompt disappears on typing which is annoying and unacceptable if a series of several variables is requested.

Input from keyboard can better be obtained via text input fields.
See subchapter 5.8 “Text Fields“ on page 34.

KEYBOARD_VISIBLE ()
returns 1 if screen keyboard is currently visible. Otherwise returns 0. This function works only when the keyboard is solid (external) and is at the bottom of the screen.

OPTION KEYBOARD OFF

OPTION KEYBOARD ON
turns ON or turns OFF physical keyboard support. If support is ON but physical keyboard is absent then software keyboard appears automatically. By default is off.

- ***Input from file***

FILE N$ INPUT X, Y$, ...
performs data input from file [n$] and stores values to specified variables [x,y$,...].
See more on input from file in chapter 6“Files & folders“ on page 37

2.3. Built-in data

DATA [list of variables]
```
DATA 1, 2, 3, "ONE", "TWO", ...
```
contains numeric or string data for using with command READ. Amount of data is not limited, data may contain expressions:
```
DATA 1+3, "One"&"num", 2, "Two"
```
Command DATA may appear more than once to store all necessary values. Values in command DATA are local, they exist only for function where they were defined.

DATA_EXIST ()
returns 1 if data when using command READ are available. Otherwise it returns 0.

READ A, B$
reads to specified variables numeric or string values, stored in command DATA.

RESTORE
resets data counter for reading values from command DATA to the beginning of data in code.

RESTORE TO _LABEL_
resets data counter for reading values from command DATA which goes in code after label [_label_].

ON X RESTORE TO LABEL1, LABEL2, ...
the same as RESTORE TO, but if [x] = 1 then resets data counter to label [label1], if [x] = 2 then resets data counter to label [label2], and so on.

2.4. Camera

ALBUM EXPORT F$
exports image or video file [f$] from smart BASIC to device camera roll.

ALBUM IMPORT F$
imports image or video file from device camera roll to smart BASIC file [f$].

Importing GIF-files

GIF files are not produced neither by iOS device nor by smart BASIC. That is why their support by ALBUM commands is not available. GIF files however are recognized and can be imported as JPG or PNG.
Notes from 'Ricardobytes' (*http://kibernetik.pro/forum/viewtopic.php?f=24&t=1585*)

- If you specify ALBUM IMPORT "file.jpg", you can import any gif file and it will be named as a jpg. If it has transparency, that will be lost. Only the first image of an AnimGif will be converted. I assumed that the gif is actually converted to a .jpg, but if I change the extender back to .gif, it can still be previewed and displayed in smart Basic programs!
- If you specify ALBUM IMPORT "file.png", you can import any gif file and it will be named as a PNG. If it has transparency, that will be retained. Only the first image of an AnimGif will be converted. I assumed that the gif is actually converted to a .png, but if I change the extender back to .gif, the image can still be previewed and displayed in smart Basic programs!
- In both cases, the images are the same size as, and indistinguishable in quality from, the original gifs.

CAMERA N$ AT BACK/FRONT PHOTO
CAMERA N$ AT BACK/FRONT VIDEO LOW/MEDIUM/HIGH
create camera [n$], connected to BACK or FRONT device camera for making PHOTO or VIDEO recordings. Camera for video recording can have optional quality parameter: LOW, MEDIUM or HIGH video quality. If parameter is not specified then low quality is used.
Examples of usage:

```
CAMERA "photocam" AT FRONT PHOTO
'creates camera for making photographs with front camera;
CAMERA "videocam" AT BACK VIDEO
'creates camera for making video recordings with back
camera;
CAMERA "videocam" AT BACK VIDEO HIGH
'creates camera for making video recordings with back
camera in high quality.
```

CAMERA N$ DELETE
deletes camera [n$].

CAMERA N$ RECORD F$
starts video recording with camera [n$] to file [f$] in MOV format.

CAMERA N$ SNAPSHOT F$
creates photo snapshot with camera [n$] and saves it to file [f$] in JPG format.
See chapter 7.3.6“Draw figures“ on page 47 for handling stored images.

CAMERA N$ STOP
stops video recording with camera [n$].

CAMERA N$ VIEW S$
displays camera [n$] inside sprite [s$].

GET CAMERA N$ SIZE W,H
gets width and height of image from camera [n$] to numeric variables [w] and [h], in points. If camera does not exist it returns [0, 0].

2.5. Phone

If the device supports making phone calls, then a phone call can be initiated in a running program:

PHONE CALL N$
performs phone call to number [n$]. Device should support making phone calls.

```
PHONE CALL "83331234567"
```

2.6. Clipboard

Smart BASIC can work with iOS clipboard, but with important differences.
First difference is capability to store not only one but multiple amount of text and numeric data in the clipboard. Usually new data replaces old data in the clipboard when iOS application saves new data to the clipboard. Smart BASIC instead appends new data to the clipboard while keeping all previous clipboard data. Order of reading data from the clipboard should be the same as order of writing data, because new data are appended to the end of the clipboard and data are read from the beginning of the clipboard.
Second difference is that data is deleted from the clipboard when it is read.
Thus clipboard may serve not only for text exchange between smart BASIC and other iOS applications, but also for data exchange between smart BASIC programs which run each other, because it can store variable amount of text and numbers.

CLIPBOARD CLEAR
clears clipboard.

CLIPBOARD READ A, B$
reads numeric or string values from the clipboard to specified variables. See clipboard specifications in preface.

CLIPBOARD WRITE A, B$
writes values of specified variables to the clipboard. See clipboard specifications in preface.

CLIPBOARD_COUNT ()
returns number of values available in the clipboard. Check this value before reading from the clipboard because attempt to read from empty clipboard rises an error.

CLIPBOARD_TYPE ()
returns type of next clipboard value available for reading:
0=undefined, 1=number, 2=string.

CLIPBOARD_TYPE (N)
returns type of [n]-th clipboard value available for reading. OPTION BASE affects order number in this function.

2.7. Print

PRINT item p item p ...
prints a string on screen, where each item is an expression and each punctuation mark p is either a comma or a semi-colon.
You can print constants and/or variables and or results of an expression.
Examples:

```
PRINT "Name ";NAME$
PRINT K$, X; Y, SQR(x^2+y^2)
```

Using "," character (comma) performs tabulated output. Using ";" character (semicolon) makes no separation between printed values (numeric values have trailing

space and may have leading space used for sign).
Each printed line is automatically finished with line feed. But if you don't need line feed at the end, you may use ";" character at the end of PRINT command. For example, these two commands will print "This is my text" in one line:

```
PRINT "This is ";
PRINT "my text"
```

If you need only line feed, you can use:

```
PRINT
```

TAB(x)
indicates output offset from beginning of line if you need custom tabulation:

```
PRINT TAB(5);TEXT$
```

TEXT CLEAR
clears text view.

- ***Format string***

If you want to use custom format output for numeric values, you can specify format string before variable using ":" separator. Format string uses "#" character for each digit, "." character for decimal dot and "E" character for exponential output. You can also use other characters in format string, and they will be present in output text.
Examples:

```
PRINT "## ###":12000
PRINT "#.##":1/3;"#.#E":1000
F$ = "#.###"
PRINT F$:X;F$:Y;F$:Z
PRINT “Large number: “;"### ### ###.##":12345*67890
```

- ***Special characters***

To print special characters like square root, you may use Unicode character codes in CHR$() function. The following code will print the square root character:

```
PRINT CHR$(8730)
```

Character codes should be given as decimal coded. To use Unicode just convert it to decimal. You can get Unicode character codes at *http://unicode-table.com*

3. Math functions

3.1. General operations and functions

Math functions in general can accept **real and complex** arguments.

Numeric math operations are: +, -, *, /, %, ^.
Operation % is a remainder of division, ^ is a power operation.

```
A = (2 ^ 3 + 1) * 2
```

Constant π
π can best be defined as constant:

```
pi=3.1415926535
```

π can also be calculated from trigonometric or complex functions, but than the angle-units should be set to radians:

```
OPTION ANGLE RADIANS
pi=ARG(-1)
```

RANDOMIZE
initializes new sequence of values for RND() and RNDC() function.

RND (X)
returns random real number from 0 to [x], not including [x].
If [x] > 1 then this is an integer number. If 0 <= [x] <= 1 then this is a float number.
Although this is a sequence of random values, it is the same at each program run.
Use RANDOMIZE command to change the sequence.

RNDC (X)
Returns random complex number with real and imaginary components from 0 to [x], not including [x]. Usage is identical to RND (X).

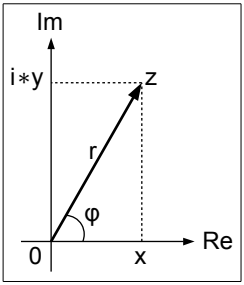
3.2. Complex numbers

A complex number is a number that can be expressed in the form $x+y*i$, where x and y are real numbers and i is the imaginary unit, that satisfies the equation i^2=−1. In the expression $x+y*i$ is x the real part and y is the imaginary part of the complex number.
In order to indicate that 'i' has to be interpreted as the imaginary unit rather than as a variable, it should be entered in Smart Basic as '1i'. In the same way any number can be made imaginary by attaching the 'i', e.g. 8i, 12.3i etc. A variable is made imaginary by multiplying it with '1i'. So the expression $z=x+i*y$ in Smart Basic should be written as $z=x+1i*y$.
A few examples with imaginary numbers and the corresponding output on screen:

```
PRINT "√-4 =";SQRT(-4)
PRINT "i^2=";1i^2
a=2+5i ! b=3+2i
PRINT "If a=";a;"and b=";b
PRINT "a+b=";a+b
PRINT "1i*a+1i*b=";1i*a+1i*b
PRINT "a*b=";a*b
```

```
√-4 = 2i
i^2=-1
If a= 2+5i and b= 3+2i
a+b= 5+7i
1i*a+1i*b=-7+5i
a*b=-4+19i
```

A complex number can be viewed as a point or position vector in a two-dimensional coordinate system called the complex plane. The numbers are conventionally plotted using the real part as the horizontal component, and imaginary part as vertical. These two values used to identify a given complex number are therefore called its Cartesian, rectangular, or algebraic form: $z=x+i*y$
This vector z can also be defined in terms of its magnitude r and direction φ relative to the origin as expressed in the following formula: $z=r*(\cos(\varphi)+i*\sin(\varphi))$
With Euler's formula $e^{i*\varphi}=\cos(\varphi)+i*\sin(\varphi)$ that leads to the complex number's polar form $z=r*e^{i*\varphi}$ ⟶
in which: r is the absolute value or norm or <u>modulus</u>
φ is the angle or <u>argument</u>
By adjusting the length r and angle φ, we can write any complex number in this way!



Multiplication by a complex number with r=1 which can be written as $z=e^{i*\varphi}$ gives a rotation by angle φ!

ARG (X)
returns argument of complex number [x].
OPTION ANGLE has its effect in this function.
With *Euler's identity* $e^{i\pi}=-1$, the value of π can be calculated as:

```
OPTION ANGLE RADIANS
pi=ARG(-1)
```

ABS(X)
If X is a complex number, then ABS(X) returns the absolute value or modulus or magnitude. For complex number $z=x+i*y$ the modulus |z| equals $|z|=\sqrt{x^2+y^2}$

IMAG (X)
returns imaginary component of complex number [x].

REAL (X)
returns real component of complex number [x].

RNDC (X)
Returns random complex number with real and imaginary components from 0 to [x], not including [x]. If [x] > 1 then this is an integer number. If 0 <= [x] <= 1 then this is a floating point number.
Although this is a sequence of random values, it is the same at each program run. Use RANDOMIZE command to change the sequence.

SGNC (X)

SIGNC (X)
return sign of imaginary part of complex number [x]: -1 if [x] < 0, 0 if [x] = 0, and 1 if [x] > 0.

Archimedean spiral in polar coordinates

An example of handling complex numbers in polar coordinates is the following code to generate an Archimedean spiral.
An Archimedean spiral describes the path of a point moving away from a fixed point with a constant angular speed. The distance r to the fixed point can be described in polar coordinates (r,φ)by the equation r=a+b*φ with real numbers a and b. The parameter 'a' determines the starting point of the spiral while 'b' controls the distance between successive turnings.

```
'Archimedean spiral, in polar coordinates
' --- constants
dphi=0.1 ' angular increment
a=0 ' starting point on real axis
b=100' amplitude parameter
GET SCREEN SIZE sw,sh
x0=sw/2 ! y0=sh/2
size=MIN(x0,y0)
' --- initiate graphics
GRAPHICS
GRAPHICS CLEAR 1,1,0.5
DRAW COLOR 1,0,0
DRAW SIZE 2
' --- initiate
DRAW TO x0+a,y0
' --- loop
DO
  r=a+b*phi
  z=r*EXP(1i*phi)
  DRAW LINE TO x0+REAL(z),y0-IMAG(z)
  phi+=dphi
UNTIL ABS(z)>size
END
```

3.3. Arithmetic

ABS (X)
returns absolute value of [x].

CEIL (X)
returns smallest integer not less than [x].

DEC (H$)
returns decimal representation of hexadecimal number, written as a string [h$]. Number is treated as unsigned integer.

```
PRINT DEC("ABCD")
```

EXP(X)
returns exponential of [x].
The exponential function is the function e^x, where e is the “Euler number” (approximately 2.718281828) with the characteristic that the function e^x is its own derivative. The function is often, as in Smart Basic, written as exp(x)

FLOOR (X)
returns largest integer not greater than [x].

FRACT (X)
returns fractional part of number [x].

HEX$ (N)
returns hexadecimal representation of integer number [n]. Number should not be negative or complex.

INT (X)
returns integer nearest to [x].
Examples for CEIL, FLOOR and INT:

```
CEIL(-1.8)=-1  FLOOR(-1.8)=-2  INT(-1.8)=-2
CEIL(-1.2)=-1  FLOOR(-1.2)=-2  INT(-1.2)=-1
CEIL( 1.2)= 2  FLOOR( 1.2)= 1  INT( 1.2)= 1
CEIL( 1.8)= 2  FLOOR( 1.8)= 1  INT( 1.8)= 2
INT(-1.5)=-2  INT( 1.5)= 2
```

INTEG (X)
returns integral part of number [x].

LN (X) or **LOG (X)**
returns natural logarithm of [x]. The natural logarithm of a number x is the power to which the “Euler number” 'e' would have to be raised to equal x. For example: log(7.389...) is 2, because e^2=7.389....

LOG (X, Y)
returns logarithm of [x] to base [y].

LOG2 (X)
returns binary logarithm of [x].

LOG10 (X)
returns decimal logarithm of [x].

MAX (X, Y)
returns maximum value of [x] and [y]. This function accepts only real arguments.

MIN (X, Y)
returns minimum value of [x] and [y]. This function accepts only real arguments.

RND (X)
returns random real number from 0 to [x], not including [x]. If [x] > 1 then this is an integer number. If 0 <= [x] <= 1 then this is a float number.
Although this is a sequence of random values, it is the same at each program run. Use RANDOMIZE command to change the sequence.

SGN (X) or **SIGN (X)**
return sign of real part of number [x]: -1 if [x] < 0, 0 if [x] = 0, 1 if [x] > 0.

SQR (X) or **SQRT (X)**
returns square root of [x].

3.4. Trigonometry

OPTION ANGLE DEGREES

OPTION ANGLE RADIANS
make all trigonometric functions to use degrees/radians as their values. Other BASIC functions and commands which use angles as their parameters are also affected. Default is radians.

ACOS (X)
returns arccosine of [x]. If argument is real then command OPTION ANGLE has its effect in this function.
Smart basic has no built-in constant for the value of π. Use e.g.:

```
PI=ACOS(-1)
```

ACOSH (X)
returns hyperbolic arccosine of [x]. If argument is real then command OPTION ANGLE has its effect in this function.

ASIN (X)
returns arcsine of [x].

ASINH (X)
returns hyperbolic arcsine of [x]. If argument is real then command OPTION ANGLE has its effect in this function.

ATN (X)

ATAN (X)
returns arctangent of [x]. If argument is real then command OPTION ANGLE has its effect in this function.

ATAN2 (Y, X)
returns arctangent of [y]/[x], from -π to +π. If arguments are real then command OPTION ANGLE has its effect in this function.

ATANH (X)
returns hyperbolic arctangent of [x]. If argument is real then command OPTION ANGLE has its effect in this function.

COS (X)
returns cosine of [x].

COSH (X)
returns hyperbolic cosine of [x].

SIN (X)
returns sine of [x].

SINH (X)
returns hyperbolic sine of [x].

TAN (X)
returns tangent of [x].

TANH (X)
returns hyperbolic tangent of [x].

3.5. Logic

AND (X, Y)
returns bitwise AND of [x] and [y].

BIT (X, N)
returns [n]-th bit of [x]. Bit numbers start from 0. If argument is complex then only real part is used.

EVEN (X)
returns 1 if real value [x] is even. Otherwise returns 0.

ODD (X)
returns 1 if real value [x] is odd. Otherwise returns 0.

OR (X, Y)
returns bitwise OR of [x] and [y].

XOR (X, Y)
returns bitwise XOR of [x] and [y].

4. String functions

ASC (A$)
returns ASCII value of first character in string [a$].

ASC (A$, N)
returns ASCII value of [n]-th character in string [a$]. Command OPTION BASE has its effect in this function.

CAPSTR$ (A$)
returns uppercase string [a$].

CHR$ (N)
converts ASCII value [n] to character.
To print special characters like square root, you may use Unicode character codes in CHR$() function. The following code will print the square root character:

```
PRINT CHR$(8730)
```

Character codes should be given decimal coded. To use Unicode just convert it to decimal. You can get Unicode character codes at *http://unicode-table.com*

INSTR (STR$, SUBSTR$, N)
returns character index in string [str$], from which string [substr$] begins. Search starts from [n]-th character of string [str$]. If string [str$] does not contain string [substr$] then returns -1. Search is case-sensitive. Command OPTION BASE has its effect in this function.

LEN (X$)
returns length of string [x$].

LEFT$ (A$, N)
returns [n] characters from the left side of string [a$].

LOWSTR$ (A$)
returns lowercase string [a$].

LTRIM$ (A$)
returns string [a$] without spaces at the left side of the string.

MID$ (A$, X)
returns substring of string [a$] which starts at index [x]. Command OPTION BASE has its effect in this function.

MID$ (A$, X, Y)
returns substring of string [a$] which starts at index [x] and has length [y]. Command OPTION BASE has its effect in this function.

MID$ (A$, X, Y, B$)
returns string [a$], in which the part, starting at index [x] and with length [y], is replaced with string [b$]. Command OPTION BASE has its effect in this function.

REVERSE$ (A$)
returns string [a$] in reverse order.

RIGHT$ (A$, N)
returns [n] characters from the right side of string [a$].

RTRIM$ (A$)
returns string [a$] without spaces at the right side of the string.

SPLIT A$ TO M$,N WITH S$
splits string [a$] to components using separator - string [s$] and stores result to one-dimensional string array [m$]. Array size is stored to numeric variable [n], if it is used. Parameter [n] is optional. Separator string [s$] contains characters, which are used to split string [a$]. Array [m$] does not contain empty strings.
Example:

```
SPLIT "1,2,3&4" TO pieces$,n WITH ",&"
```

SPLITE A$ TO M$,N WITH S$
(SPLIT Empty) the same as SPLIT command, but array [m$] can contain empty strings.

STR$ (N, F$)
converts number [n] to string using format string [f$]. Format string is the same as in PRINT command.
Example in Graphics mode:

```
DRAW TEXT STR$(12345*67890,"### ### ###.##") AT 0,0
```

SUBSTR$ (A$, X, Y)
returns substring of string [a$] which starts at index [x] and ends at index [y]. Command OPTION BASE has its effect in this function.

TRIM$ (A$)
returns string [a$] without spaces at both sides of the stringSTR$ (N)
converts number [n] to string. Note that in smart BASIC number to string and string to number conversion is performed automatically where it is possible.

UNIQUE_STR$ ()
returns unique string. One of possible implementations: for generating temporary

file names which should not accidentally coincide with any of existing file names.

VAL (X$)
converts string [x$] to number. Note that in smart BASIC number to string and string to number conversion is performed automatically where it is possible.

5. Interface objects

Interface objects are the communication tool between a user and an app. They convey a particular action or intention to the app through user interaction, and can be used to manipulate content, provide user input and output, navigate within an app, or execute other pre-defined actions.
The available objects in Smart Basic are:

- Buttons: activate actions by a tap
- Switches: lets the user turn an option on and off
- Sliders: enable users to interactively modify some adjustable value
- List panels: for selecting a specific item from a list
- Text Fields: allows input or output of a single or multiple lines of text
- Browsers: allows output in HTML format

About object creation

Interface object name is a string and it cannot be empty - it is used in commands and functions to identify this specific object by its name. Object name is case sensitive.
It is correct to re-create object with the same name if you need to change some of its attributes. But special commands often exist for this purpose. For example, to change text in text input field you can use command FIELD SET TEXT:

```
FIELD n$ SET TEXT "New text"
```

5.1. About pages

Pages are nameable, moveable, resizeable zones for interface objects grouping and placement.
Every created interface object belongs to currently active page. Pages can be created and manipulated by PAGE commands. Default page has empty name "". This default page will not be created until it is needed. A new page is default created with transparent background and fullscreen size. Size and color can be customized.
The active page covers other pages which are located below. To copy text from text window or to get info from buttons, lists or text fields, you will need to clear access to it by hiding all blocking pages, even if their background is transparent.
The program 'Panel demo' on the forum gives an example of several pages used for graphics, text, lists and buttons displayed simultaneously.
You'll find it at: *http://kibernetik.pro/forum/viewtopic.php?f=20&t=811*

PAGE N$ ALPHA X
sets alpha of page [n$] to value [x]. Valid values are from 0 to 1.

PAGE N$ AT X,Y
sets coordinates of page [n$] to point [x,y].

PAGE N$ COLOR R,G,B,A
sets color of page [n$] to value with red [r], green [g], blue [b] and alpha [a] components. Valid values are from 0 to 1.

PAGE N$ FRAME X,Y, W,H
sets coordinates of page [n$] to point [x,y], width to value [w] and height to value [h].

PAGE N$ HIDE
hides page which name is [n$].

PAGE N$ SET
makes page [n$] active. If such page does not exist then it is created.

PAGE N$ SHOW
shows previously hidden page which name is [n$].

5.2. Presets

SET BUTTONS CUSTOM
SET BUTTONS DEFAULT
Set newly created buttons to be of "custom" or "default" type.
Custom button type means that when button is created its title color and opacity are defined by DRAW COLOR and alpha, also its background color and opacity are defined by FILL COLOR and alpha.
Default button type means that buttons have standard appearance.
The button-size WxH is: (20+text-length)x(12+text-height)

SET BUTTONS FONT DEFAULT
sets font name and size of newly created buttons to default values.

SET BUTTONS FONT NAME N$
sets font name of newly created buttons to [n$].

SET BUTTONS FONT SIZE X
sets font size of newly created buttons to [x].

SET LISTS CUSTOM

SET LISTS DEFAULT
set newly created lists to be of "CUSTOM" or "DEFAULT" type.
Custom list type means that when list is created its text color and opacity are defined by DRAW COLOR and alpha, also its background color and opacity are defined by FILL COLOR and alpha.
Default list type means that lists have standard appearance.

SET LISTS FONT DEFAULT
sets font name and size of newly created lists to default values.

SET LISTS FONT NAME N$
sets font name of newly created lists to [n$].

SET LISTS FONT SIZE X
sets font size of newly created lists to [x].

SET TOOLBAR OFF
SET TOOLBAR ON
turns OFF and turns ON visibility of top control toolbar. It also affects position of main graphics window on the screen.

TOOLBAR_VISIBLE ()
returns 1 if top control toolbar is visible. Otherwise it returns 0.

5.3. Handling existing objects

5.3.1 Listings of available objects

LIST BUTTONS TO A$,N
saves list of existing buttons' names to string array [a$] and size of returned array to numeric variable [n].

LIST FIELDS TO A$,N
saves list of existing text fields' names to string array [a$] and size of returned array to numeric variable [n].

LIST LISTS TO A$,N
saves list of existing lists' names to string array [a$] and size of returned array to numeric variable [n].

LIST SLIDERS TO A$,N
saves list of existing sliders' names to string array [a$] and size of returned array to numeric variable [n].

LIST SWITCHES TO A$,N
saves list of existing switches' names to string array [a$] and size of returned array to numeric variable [n].

5.3.2 Changing and testing visibility

BUTTON N$ HIDE
hides button which name is [n$].

BUTTON N$ SHOW
shows previously hidden button which name is [n$].

BUTTON_VISIBLE (N$)
returns 1 if button with name [n$] is shown, otherwise it returns 0.

FIELD N$ DESELECT
deactivates text input field which name is [n$].

FIELD N$ HIDE
hides text field which name is [n$].

FIELD N$ SELECT
activates text input field which name is [n$]. If text input field contains text then this text will be selected.

FIELD N$ SELECT X
it positions cursor in text input field which name is [n$] after [x] characters.

FIELD N$ SELECT X,Y
it positions cursor in text input field which name is [n$] after [x] characters and selects [y] characters.

FIELD N$ SHOW
shows previously hidden text input field which name is [n$].

FIELD_VISIBLE (N$)
returns 1 if text input field with name [n$] is shown, otherwise it returns 0.

LIST N$ HIDE
hides list which name is [n$].

LIST N$ SHOW
shows previously hidden list which name is [n$].

LIST_VISIBLE (N$)
returns 1 if list with name [n$] is shown, otherwise it returns 0.

SLIDER N$ HIDE
hides slider which name is [n$].

SLIDER N$ SHOW
shows previously hidden slider which name is [n$].

SLIDER_VISIBLE (N$)
returns 1 if slider with name [n$] is shown, otherwise it returns 0.

SWITCH N$ HIDE
hides switch which name is [n$].

SWITCH N$ SHOW
shows previously hidden switch which name is [n$].

SWITCH_VISIBLE (N$)
returns 1 if switch with name [n$] is shown, otherwise it returns 0.

TOOLBAR_VISIBLE ()
returns 1 if top control toolbar is visible. Otherwise it returns 0.

5.3.3 About sizing for different devices

Info from the forum:

> by **ricardobytes** » Sun Jun 05, 2016
> To explain about my screen sizing code. It uses my iPad Air as the standard screen size, 1024 x 768 points, since I do all my development on it.
> The first thing I do is get the screen width *sw* and screen height *sh* of the device that is running the code.
> The variables *ratw* and *rath* are short for "ratiowidth" and "ratioheight" and are calculated by dividing the current device's width and height by the iPad's width and height. These are usually fractions, since the screens of most other iOS devices (iPhones and iPods) will be smaller.
> Then the sizes and positions of every graphic or interface object are multiplied by these two ratios, and usually the resulting display will fit nicely on the smaller device. Adjusting font sizes can be tricky. I find that in most cases, multiplying the font size by *ratw* usually works.

About sizing of buttons

From the forum:

> *Actually automatic button size equals: (20 + text_length) x (12 + text_height)*
> *Field does not resize automatically after text or font size is changed.*
> *Automatic size is set only when field is created and field size is not specified.*

Resizing after changing text or font size could be done as in the following sample code:

```
f$="Helvetica"
fs=30
t$="This is my text"
DRAW FONT NAME f$
DRAW FONT SIZE fs
```

```
w=TEXT_WIDTH(t$)+16 'offset 14 is minimum width 'border'
h=TEXT_HEIGHT(t$)+8 'offset 8 can be reduced to zero
FIELD 0 AT 10,50 SIZE w,h
FIELD 0 FONT NAME f$
FIELD 0 FONT SIZE fs
FIELD 0 TEXT t$
End 'of sample code
```

5.4. Buttons

BUTTON N$ DELETE
Deletes button which name is [n$].

BUTTON N$ SET TEXT T$
sets text [t$] for button with name [n$].

BUTTON N$ TEXT T$ AT X,Y SIZE W,H
creates button with name [n$] and text [t$] at point [x,y] with width [w] and height [h]. Parameter SIZE is optional, if it is not set then button is autosized.

BUTTON_PRESSED (N$)
Returns 1 if button with name [n$] was pressed, otherwise it returns 0.

SET BUTTONS CUSTOM

SET BUTTONS DEFAULT
Set newly created buttons to be of "custom" or "default" type.
Custom button type means that its title color and opacity are defined by draw color and alpha, also its background color and opacity are defined by fill color and alpha.
Default button type means that buttons have standard appearance.

5.5. Switches

SWITCH N$ DELETE
deletes switch which name is [n$].

SWITCH N$ SET STATE K
sets state [k] for switch with name [n$]. If [k] = 0 then switch is off, otherwise it is on.

SWITCH N$ STATE K AT X,Y
creates switch with name [n$] at point [x,y] with state [k]. If [k] = 0 then switch is off, otherwise it is on.

SWITCH_CHANGED (N$)
returns 1 if state of switch with name [n$] was changed. Otherwise it returns 0.

SWITCH_STATE (N$)
returns state of switch with name [n$].
On = 1, off = 0.

5.6. Sliders

SLIDER N$ DELETE
deletes slider with name [n$].

SLIDER N$ SET VALUE K
sets value [k] for slider with name [n$]. Valid values for [k] are from 0 to 1.

SLIDER N$ VALUE K AT X,Y SIZE S ANGLE A
creates slider with name [n$], value [k], at point [x,y], with size [s] and at an angle [a] to horizontal direction. Valid values for [k] are from 0 to 1. Parameter ANGLE is

optional. Command OPTION ANGLE affects this command.

SLIDER_CHANGED (N$)
returns 1 if value of slider with name [n$] was changed. Otherwise it returns 0.

SLIDER_VALUE (N$)
returns value of slider with name [n$].

5.7. List panels

LIST N$ DELETE
deletes list with name [n$].

LIST N$ HIDE
hides list which name is [n$].

LIST N$ SELECT K
selects row with number [k] in list with name [n$]. To clear selection, set row number to -1. OPTION BASE command affects this command.

LIST N$ SHOW
shows previously hidden list which name is [n$].

LIST N$ TEXT M
sets contents of list with name [n$] equal to contents of one-dimensional array [m].

LIST N$ TEXT M AT X,Y SIZE W,H
creates list with name [n$], with contents of one-dimensional array [m], at point [x,y] and with size [w,h].
SET LISTS command affects appearance of newly created lists.

LIST_SELECTED (N$)
returns number of selected row in list with name [n$]. If nothing is selected then returns -1. OPTION BASE command affects this function.

SET LISTS CUSTOM

SET LISTS DEFAULT
set newly created lists to be of "CUSTOM" or "DEFAULT" type.
Custom list type means that when list is created its text color and opacity are defined by draw color and alpha, also its background color and opacity are defined by fill color and alpha.
Default list type means that lists have standard appearance.

5.8. Text Fields

Text fields can be used for input and output of text.

Some notes on user interface with text input fields

User interface with the INPUT command (in text view) is rather clumsy and primitive. See “User interface with INPUT command ” on page 18.
The usage of text input fields gives a good alternative:

In Smart Basic no function is available to determine the current screen-position in text view. In textfields however the cursor-position can be derived from the command FIELD_CURSOR_POS(). Using that command an input-field can be realised in which the prompt is 'safe', i.e. it can not be overwritten.

The following code gives an example of input-fields with 'safe' prompts:

```
'Inline Input Field
n=1
DO
```

```
 Prompt$="Request "&STR$(n,"#")&": "
 Field$="In_"&STR$(n,"#")
 Answer$=Input$(10,75+n*25,400,22,Field$,Prompt$)
 PRINT "Input";n;"=";Answer$
n+=1
until n>3 OR Answer$=""
END

DEF Input$(x,y,w,h,Field$,Prompt$)
FIELD Field$ AT x,y SIZE w,h
FIELD Field$ FONT SIZE h-6
CALL InputPreset(Field$)
FIELD Field$ SELECT
prompt=LEN(Prompt$)
DO
 IF FIELD_CURSOR_POS(Field$)<prompt THEN FIELD Field$ TEXT
Prompt$
 SLOWDOWN
UNTIL FIELD_CHANGED(Field$)
Txt$=FIELD_TEXT$(Field$)
T$=RIGHT$(Txt$,LEN(Txt$)-prompt)
RETURN T$
END DEF

DEF InputPreset(name$)
FIELD name$ BACK COLOR 0,1,0
FIELD name$ FONT NAME "Verdana"
FIELD name$ FONT COLOR 0,0,1
END DEF
```

FIELD N$ BACK ALPHA X
sets background alpha of text field which name is [n$] to value [x]. Valid values are from 0 to 1.

FIELD N$ BACK COLOR R,G,B
sets background color of text field which name is [n$] to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

FIELD N$ DELETE
deletes text field which name is [n$].

FIELD N$ DESELECT
deactivates text field which name is [n$].

FIELD N$ FONT ALPHA X
sets font alpha of text field with name [n$] to value [x]. Valid values are from 0 to 1.

FIELD N$ FONT COLOR R,G,B
sets font color of text field which name is [n$] to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

FIELD N$ FONT NAME T$
sets font name of text field which name is [n$] to value [t$].

FIELD N$ FONT SIZE S
sets font size of text field which name is [n$] to value [s].

FIELD N$ SELECT
activates text input field which name is [n$]. If text input field contains text then this

text will be selected.

FIELD N$ SHOW
shows previously hidden text field which name is [n$].

FIELD N$ TEXT A$
sets text [a$] for text field with name [n$].

FIELD N$ TEXT A$ AT X,Y SIZE W,H ML RO
creates text field with name [n$] and text [a$] at point [x,y] with width [w] and height [h]. Parameter ML (multi-line) is optional, if it is used then text field will be multi-line. Parameter RO (read-only) is optional, if it is used then text field is read-only. Parameter TEXT is optional, if it is not set then text field is created empty. Parameter SIZE is optional, if it is not set then text field is autosized.

FIELD_CHANGED (N$)
returns 1 if user pressed Enter in input field with name [n$], otherwise it returns 0.

FIELD_CURSOR_POS (N$)
returns position of cursor in text input field with name [n$].

FIELD_CURSOR_SEL (N$)
returns number of selected characters in text input field with name [n$].

FIELD_TEXT$ (N$)
returns text of input field with name [n$].

5.9. Browsers

Browser commands refer to the current page. By page usage, several browsers can be created and hidden or displayed.
HTTP communication is described in chapter 10“Networking” on page 65.

BROWSER N$ DELETE
deletes browser [n$].

BROWSER N$ URL A$ AT X,Y SIZE W,H
creates browser with name [n$] at point [x,y] with width [w], height [h] and loads web page with URL [a$]. Parameter URL is optional.
Example:

```
BROWSER "n" URL "http://kibernetik.pro" AT 0,0 SIZE
SCREEN_WIDTH(),SCREEN_HEIGHT()
```

HTML format-tags can be used to format text in the browser.
Example:

```
top$="<center><h3>Text Panel</h3></center>"
msg1$="Some text<br><br>Written with "
msg2$="<b>HTML</b> format and styling tags"
BROWSER “panel” SET TEXT top$&msg1$&msg2$
```

Info on tags can be found at http://www.w3schools.com/tags/

BROWSER N$ DELETE
deletes browser [n$].

BROWSER N$ TEXT T$ URL A$
sets contents of web page in browser [n$] as string [t$]. Parameter URL is optional, it is used to specify URL of web page if its contents is address-dependent.
This command, without parameter URL, can be used to display text (JavaScript string) in a browser 'window' for e.g. messages or help-info. Example:

```
graphics
```

```
graphics clear
browser 0 at 30,30 size 120,300
t$="My browser<br>with <b>multi-line text</b>"
browser 0 set text t$
```

BROWSER N$ URL A$
loads web page with URL [a$] in browser [n$].

BROWSER N$ URL A$ AT X,Y SIZE W,H
creates browser with name [n$] at point [x,y] with width [w], height [h] and loads web page with URL [a$]. Parameter URL is optional.
Example:

```
BROWSER "n" URL "http://kibernetik.pro" AT 0,0 SIZE
SCREEN_WIDTH(),SCREEN_HEIGHT()
```

BROWSER_TEXT$ (N$, T$)
returns contents of current web page in browser [n$] using JavaScript string [t$].
Example:

```
http://kibernetik.pro/forum/viewtopic.php?f=20&p=11291BROWSER "n" URL
"http://apple.com" AT 0,0 SIZE 0,0
PRINT "URL: " & BROWSER_TEXT$("n", "document.baseURI")
PRINT "Number of images: " & BROWSER_TEXT$("n",
"document.images.length")
PRINT "First image: " & BROWSER_TEXT$("n",
"document.images[0].src")
```

SET BROWSERS NORMAL
SET BROWSERS SCALED
set page display mode for newly created browsers. If NORMAL then browser loads page non-scaled, if SCALED then browser scales loaded page to fit browser window.

About the size of fields and browsers
Input from the forum, topic about the app '**Prompter**'
http://kibernetik.pro/forum/viewtopic.php?f=20&p=11291

> by **rbytes**, May 2017
> … Even though the screen is of a fixed dimension for each device (e.g. my iPad has 1024 x 768 points or 2048 x 1536 pixels), when you create certain interface objects, they can be much taller than the screen. So far I have tested pages, fields and browsers. I haven't found the absolute limit yet, but so far I have created interface objects 20,000 points high! Then by varying the y value for the top of the object using a loop, I can scroll it upward to view all of it on the screen. …
>
> … I have now tested creating a super-wide page and browser and scrolling them sideways. …
>
> … I have tested it with a page and browser 26 times the width of the screen and they work fine. There is a slight delay and flash when they are created, but they are stable after that.

6. Files & folders

File commands can be divided in two sets: simple and advanced.
Simple commands are FILE PRINT, FILE INPUT and FILE RESET. They are similar to PRINT, DATA, READ and RESTORE commands, but using files. They perform

text data output to file and reading data from file. Simple commands do not change file pointer and do not depend on file pointer.
All other file commands and functions belong to advanced set. They perform byte level access to files, they use file pointers, can detect end of file and have other possibilities.
File pointer changes its position automatically, you do not need to change it manually after each file command.

6.1. Current, parent and root directory

File names are relative to current directory.
If you need to specify current directory, you may use empty name "" or single dot name ".". If you need to specify parent directory you may use double dot name "..". If you need to specify root directory you may use "/".
Examples:

```
DIR "" LIST FILES a$,b
DIR "." LIST FILES a$,b
DIR ".." LIST FILES a$,b
DIR "/" LIST FILES a$,b
```

Current directory is initially set to a folder where you run your program, but it can be changed with command DIR SET.

CURRENT_DIR$ ()
returns current directory.

6.2. File writing options

OPTION FILEMODE INSERT

OPTION FILEMODE OVER
Set file writing functions into insert or overwrite mode. This affects how data is written to the file. In insert mode file writing functions insert new data at position of file pointer, shifting existing data so nothing is overwritten. In overwrite mode file writing functions overwrite existing data. Default is overwrite mode.

6.3. Directory commands and functions

CURRENT_DIR$ ()
returns current directory.

DIR N$ COPY M$
copies directory [n$] with all its contents to folder [m$].

DIR N$ CREATE
creates directory [n$]. If necessary, all intermediate folders are also created.

DIR N$ DELETE
deletes directory [n$] with all its contents. You cannot delete current directory, but if you want to delete it then change current directory and specify proper name of directory to be deleted.

DIR N$ LIST DIRS A$, B
saves list of folder names in directory [n$] to single-dimensional string array [a$]. Number of folders is saved to numeric variable [b].
List of folders is alphabetically sorted.
If array [a$] was declared earlier then its size is changed (increased or decreased) to store list of folders.

If array was not declared before then it is automatically created with size equal to number of folders. Size of array [a$] is stored to variable [b].

DIR N$ LIST FILES A$, B
saves list of file names in directory [n$] to single-dimensional string array [a$]. Number of files is saved to numeric variable [b].
List of files is alphabetically sorted.
If array [a$] was declared earlier then its size is changed (increased or decreased) to store list of files.
If array was not declared before then it is automatically created with size equal to number of files. Size of array [a$] is stored to variable [b].

DIR N$ RENAME M$
renames directory [n$] to folder [m$].

DIR N$ SET
sets current directory to [n$].

6.4. File commands and functions

DATA_EXIST (N$)
returns 1 if data in file [n$] when using command FILE INPUT are available. Otherwise it returns 0.

FILE N$ APPEND D$
appends contents of file [d$] to the end of file [n$]. This command does not depend on current file pointers and does not change them.

FILE N$ COPY D$
copies specified file [n$] to file [d$].

FILE N$ DELETE
deletes specified file [n$].

FILE N$ INPUT X, Y$, ...
performs text data input from file [n$] and stores values to specified variables [x,y$,...]. This command is similar to READ command, but using data contained in file. This command does not change file pointer and also does not depend on position of file pointer. Resetting input to the start of file should be done with the command FILE N$ RESET. Function DATA_EXIST(n$) may be useful when using this command. Text data in file should be separated by spaces or tabs. Complex strings should be used in quotes "".

FILE N$ PRINT ...
prints specified data into file [n$]. Parameters [...] are similar to parameters of PRINT command.
This command does not change file pointer and also does not depend on position of file pointer.
This command is similar to PRINT command, but using file for text output. It always adds text to the end of file.

FILE N$ READ X
reads byte (value 0..255) from file [n$] and stores it to variable [x]. If reading after end of file was reached, variable receives value of -1.
It is possible to specify several variables to read multiple bytes:

```
FILE n$ READ x,y,...
```

FILE N$ READDIM M
reads bytes in file [n$] to one-dimensional numeric array [m]. Size of array [m] changes to number of read bytes.

FILE N$ READDIM M, N
reads bytes in file [n$] to one-dimensional numeric array [m], and number of read bytes is stored to numeric variable [n].
Size of array [m] changes to number of read bytes.

FILE N$ READDIM M, N, K
reads [k] bytes from file [n$] to one-dimensional numeric array [m], and number of actually read bytes is stored to numeric variable [n]. Size of array [m] changes to number of read bytes.

FILE N$ READLINE X$
reads string line from file [n$] and stores it to variable [x$]. Carriage return is not stored in the string. It is possible to specify several variables to read multiple string lines:

```
FILE n$ READLINE x$,y$,...
```

FILE N$ RENAME D$
renames specified file [n$] to new name [d$]. File pointer of file with new name is retained from file with old name.

FILE N$ RESET
resets file [n$]. After reset, FILE INPUT command starts reading data from the beginning of file.
This command does not change file pointer. FILES RESET command is for resetting FILES INPUT command only.
This command is similar to RESTORE command, but using data contained in file.

FILE N$ SETPOS X
sets current position of file pointer for file [n$] to value [x].
If the filepointer x in "FILE f$ SETPOS x" is past the end of the actual file size, then the filepointer is set to the end of the file.

FILE N$ TRIM
deletes all data after current position of file pointer in file [n$]. It is possible to specify file pointer value [x] after which data will be deleted:

```
FILE n$ TRIM x
```

FILE N$ WRITE X
writes byte (value 0..255) from variable [x] to file [n$].
It is possible to specify several variables to write multiple bytes:

```
FILE n$ WRITE x,y,...
```

FILE N$ WRITEDIM M
writes contents of one-dimensional numeric array [m] as bytes to file [n$]. Array [m] must contain only values from 0 to 255.

FILE N$ WRITEDIM M, N
writes [n] elements of one-dimensional numeric array [m] as bytes to file [n$]. Array [m] must contain only values from 0 to 255.

FILE N$ WRITEDIM M, N, K
writes [n] elements of one-dimensional numeric array [m] starting with element number [k] as bytes to file [n$]. Array [m] must contain only values from 0 to 255. Com-

mand OPTION BASE has its effect on this command.

FILE N$ WRITELINE X$
writes string from variable [x$] to file [n$]. Carriage return is automatically added to the string.
It is possible to specify several variables to write multiple string lines:

```
FILE n$ WRITELINE x$,y$,...
```

If you need to write string line without adding carriage return at the end, you can use ";" separator after variable, for example:

```
FILE n$ WRITELINE x$;
FILE n$ WRITELINE x$;y$;
```

FILE_END (N$)
returns 1 if file pointer of file [n$] is currently at the end of file. Otherwise it returns 0.

FILE_EXISTS (N$)
returns 1 if file or directory with name [n$] exists. Otherwise it returns 0.

FILE_POS (N$)
returns current position of file pointer for file [n$].

FILE_SIZE (N$)
returns size of file [n$], in bytes. On error returns -1.

6.5. Compression and decompression

GUNZIP Z$ TO F$
unpacks file [z$], which was compressed with DEFLATE algorithm, and saves result to file [f$].

GZIP F$ TO Z$
packs file [f$] using DEFLATE algorithm and saves result to file [z$].

6.6. Cloud storage in Dropbox

Smart BASIC is designed to work with Dropbox cloud storage. By default this option is disabled but it can be turned on with command OPTION DROPBOX ON .
After executing this command, new icon is added to smart BASIC file interface at the bottom right. You can access Dropbox storage using this icon. Of course you will need Dropbox account, which can be obtained at website *www.dropbox.com*.
When you tap Dropbox icon (bottom right icon) in the files interface display, then you'll enter Dropbox. The arrow will initially be pointing to the cloud and after tapping the arrow will be pointing from the cloud to local.
Here you can select/copy/paste files you need. Then with the same button you can return to Local files storage to select/copy/paste files.

Apple does not allow the Dropbox facility to copy files with extension '.txt' from Dropbox. Smart Basic however allows to edit and run program-files with other (although not all) extensions. E.g. the extension '.sb', initiated by Microsoft for 'Small Basic', can be used instead.
The default extension '.txt' for program-files can be overruled by naming or renaming. Folders containing '.txt'-files can be copied from Dropbox.

OPTION DROPBOX OFF
OPTION DROPBOX ON
disables/enables Dropbox integration. Dropbox storage is used for backup and to copy files between your computer and smart BASIC. You cannot run programs from

remote file system directly. Default is disabled.
Note: you cannot copy program files *.TXT from Dropbox to the device.

7. Display on screen

For display on screen a choice has to be made between text view or graphics view. Text view is the default mode. In general its use will be limited to the display of text handling or the result of calculations. Contrary with graphics view however, it doesn't need preset of display options and is ready for use.

7.1. Screen characteristics

SCREEN_HEIGHT ()
returns screen height for current device orientation.

SCREEN_SCALE ()
returns screen scale. Non-Retina screens have screen scale 1. Retina screens have screen scale 2.

SCREEN_WIDTH ()
returns screen width for current device orientation.
The following code displays the screen characteristics.

```
PRINT "Screen WxH =";SCREEN_WIDTH();"x";SCREEN_HEIGHT()
PRINT "Retina: ";
IF SCREEN_SCALE()=2 THEN
  PRINT "Yes"
ELSE
  PRINT "No"
ENDIF
```

GET SCREEN SIZE W,H
gets width and height of the screen for current device orientation to variables [w] and [h], in points.

7.2. Text view

Text-view is the default mode of Smart Basic. Except for font and color the output style can not be chosen.
Free positioning of text in horizontal and/or vertical screen-coordinates is not possible. Interface objects (chapter 5), custom made “pages” (subchapter 5.1) and sprites (chapter 8) however, can be positioned freely.
Text input from the keyboard either virtual or external via bluetooth, via the command INPUT, is default in a text-field at the top of the screen.

TEXT
switches from graphics to text view.

TEXT CLEAR
clears text view.

7.2.1 Text output styling

FONT F$ LOAD N$
loads TTF-font from file [f$]. Font name is saved to string variable [n$].

SET OUTPUT BACK COLOR R,G,B
sets text output window background color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

SET OUTPUT FONT COLOR R,G,B
sets text output window font color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

SET OUTPUT FONT NAME N$
sets name of text output window font to [n$]. List of fonts you can get by command LIST FONTS.

SET OUTPUT FONT SIZE N
sets size of text output window font to value [n].

Fit font-size to screen-width
with monospaced fonts, e.g. “Courier” or "Menlo-Regular", the number of characters per line can be preset with the following code:

```
GET SCREEN SIZE sw,sh
GET SCREEN SIZE sw,sh
fmin=10 'minimum fontsize
fsize=MAX((sw-20)/(0.6*n),fmin)
SET OUTPUT FONT SIZE fsize
```

'Emoji'-characters can also be printed but need special attention.
See posted item on the forum at *http://bit.ly/1P6Akrx*

7.3. Graphics view

In graphics view full control of each display-pixel is possible.

GRAPHICS
switches to graphics view.

SET TOOLBAR OFF
SET TOOLBAR ON
turns OFF and turns ON visibility of top control toolbar. It also affects position of main graphics window on the screen.

TOOLBAR_VISIBLE ()
returns 1 if top control toolbar is visible. Otherwise it returns 0.

7.3.1 Handling Retina display resolution

In smart BASIC graphics commands are usually Retina-independent, so drawing line from 0,0 to 100,100 will look similar on Retina and on non-Retina screen. But pixel-level commands (for example DRAW PIXEL) are Retina-dependent, because they have access to physical pixels.Screen scale value returned by function SCREEN_SCALE() shows whether screen is Retina or not.

7.3.2 Presets

DRAW ALPHA X
sets draw alpha to value [x]. Valid values are from 0 to 1.
Alpha determines the transparency of the graphics.
In computer graphics, alpha compositing is the process of combining an image with a background to create the appearance of partial or full transparency.

DRAW COLOR R,G,B
sets draw color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

DRAW DASH X
sets dashed line with interval [x]. Value 0 sets solid line. Example:
```
DRAW DASH 10
```
You can also use custom dash pattern by specifying multiple values indicating solid and space intervals.
Example:
```
DRAW DASH 20,10,1,10
```
If you set line style, then it continues to all next drawing lines.
The command DRAW DASH 0 turns it off.

DRAW DASH X PHASE Y
Parameter PHASE sets offset of dash pattern from the beginning of line for [y] points. By drawing a line with pattern X and cycling phase Y, a 'moving' line pattern can be realized. If the PHASE parameter 'Y' is made negative, the direction of motion will reverse. If the PHASE parameter is not made cyclic, but steadily increasing or decreasing, then the cycle-period will automatically be made equal to the length of the pattern. See 'Moving lines' example attached to the command REFRESH.

DRAW FONT NAME N$
sets graphics font name to [n$]. Default is "Courier-Bold".

DRAW FONT SIZE X
sets graphics font size to [x]. Default is 20.

FONT F$ LOAD N$
loads TTF-font from file [f$]. Font name is saved to string variable [n$].

FONT_SIZE ()
returns graphics font size.

DRAW LINECAP RECT

DRAW LINECAP ROUND
sets style of line end: RECT - rectangular, ROUND - rounded. Default is RECT.

DRAW SIZE X
sets pen thickness to value [x].
In general the size will be an integer value. Decimal fractions however will also work. Minimum physical size is one pixel, but in case of x<1 the subjective size will be decreased by lowering the intensity.
See *http://kibernetik.pro/forum/viewtopic.php?f=20&t=1563*

DRAW TO X,Y
sets current pen coordinates to [x,y].

FILL ALPHA X
sets fill alpha to value [x]. Valid values are from 0 to 1.

FILL COLOR R,G,B
sets fill color to value [r,g,b] with red, green and blue components. Valid values are from 0 to 1.

GRAPHICS
switches to graphics view.

GRAPHICS CLEAR
clears graphics view with black color.

GRAPHICS CLEAR R,G,B
clears graphics view with specified [r,g,b] color of red, green and blue components. Valid values are from 0 to 1.

GRAPHICS CLEAR R,G,B,A
clears graphics view with specified [r,g,b] color of red, green, blue components and value of alpha-channel [a]. Valid values are from 0 to 1.

GRAPHICS MODE X
sets graphics commands compositing mode to X, where X is one of the modes listed below.
Available modes are:
NORMAL, MULTIPLY, SCREEN, OVERLAY, DARKEN, LIGHTEN, COLORDODGE, COLORBURN, SOFTLIGHT, HARDLIGHT, DIFFERENCE, EXCLUSION, HUE, SATURATION, COLOR, LUMINOSITY, CLEAR, COPY, SOURCEIN, SOURCEOUT, SOURCEATOP, DESTOVER, DESTIN, DESTOUT, DESTATOP, XOR, PLUSDARKER, PLUSLIGHTER
Default mode is "NORMAL".
Example:

```
GRAPHICS
FILL RECT 50,50 SIZE 50
GRAPHICS MODE CLEAR
FILL RECT 50,50 SIZE 25
```

A short description of the several modes is given in the documentation-section of the app, (in editor, touch symbol 🕮)
A more detailed description with images can be found in the *Apple developers lib*
From that info, three documents have been extracted and stored In the DropBox-folder of this manual *http://bit.ly/1DN8gYw*
These documents are:
“Image composition modes”, “Color and color spaces” and “Transparency layers”

OPTION IMAGE POS CENTRAL
OPTION IMAGE POS NORMAL
set coordinate mode for DRAW IMAGE command: "CENTRAL" - coordinates set image center; "NORMAL" - coordinates set image left corner. Default is NORMAL.

OPTION TEXT POS CENTRAL
OPTION TEXT POS NORMAL
set coordinate mode for DRAW TEXT command: "CENTRAL" - coordinates set text center; "NORMAL" - coordinates set text left corner. Default is NORMAL.

REFRESH
updates graphics window.

REFRESH OFF
turns off automatic graphics window update.

REFRESH ON
turns on automatic graphics window update. By default updating is on.
REFRESH OFF disables automatic screen update and REFRESH ON enables it and forces immediate update. The command REFRESH just updates screen without turning on automatic updates. It allows you to update graphics screen exactly when you need to, without turning on/off automatic screen updates.

Example:

```
' moving lines
' with pattern length 45
GRAPHICS
sw=SCREEN_WIDTH()
sh=SCREEN_HEIGHT()
DRAW SIZE 5
REFRESH OFF
DO
  i+=1
  DRAW DASH 20,10,5,10 PHASE i*5
  GRAPHICS CLEAR 0,0,0
  DRAW CIRCLE sw/2,sh/2 SIZE sw/4
  DRAW RECT 2,2 TO sw/3,sh/3
  DRAW LINE 2,sh/2 TO sw,sh
  REFRESH
  i%=9
  GET TOUCH n AS x,y
UNTIL x>-1 AND y>-1
END
```

SHADOW ALPHA X
sets shadow opacity to value [x]. Valid values are from 0 to 1. Default is 1/3.

SHADOW BLUR X
sets shadow blur to value [x]. Default is 2.

SHADOW COLOR R,G,B
sets shadow color to value [r,g,b] with red, green and blue components. Valid values are from 0 to 1. Default is [0,0,0].

SHADOW OFF
turns shadows off.

SHADOW OFFSET X,Y
sets shadow offset in both directions. Default is [3,3].

SHADOW ON
turns shadows on.

TEXT
switches from graphics to text view.

7.3.3 Draw text

Note on Emoji-characters
'Emoji'-characters can also be drawn but need special attention.
See posted item on the forum at *http://bit.ly/1P6Akrx*

DRAW TEXT T$ AT X,Y
draws text [t$] at point [x,y] which indicates top left corner of text.
OPTION TEXT POS command affects this command.

DRAW FONT NAME N$
sets graphics font name to [n$]. Default is "Courier-Bold".

DRAW FONT SIZE X
sets graphics font size to [x]. Default is 20.

FONT F$ LOAD N$
loads TTF-font from file [f$]. Font name is saved to string variable [n$].

FONT_SIZE ()
returns graphics font size.

TEXT_HEIGHT (T$)

TEXT_WIDTH (T$)
return height and width of text [t$] for current graphics font.

7.3.4 Draw pixels

DRAW PIXEL X,Y COLOR R,G,B,A
draws pixel at coordinates [x,y] with color of red [r], green [g] and blue [b] components, and with alpha [a]. Parameter COLOR can be omitted to use current draw color and alpha:

```
DRAW PIXEL x,y
```

Parameter [a] can be omitted. In this case alpha is equal to 1:

```
DRAW PIXEL x,y COLOR r,g,b
```

This command is screen scale-dependent.

GET PIXEL X,Y COLOR R,G,B,A
gets color of pixel at coordinates [x,y] to variables for red [r], green [g] and blue [b] components, and alpha-value to variable [a]. This command is screen scale-dependent.

> *DRAW PIXEL and GET PIXEL commands are **NOT fully reversible**! Smart BASIC uses premultiplied alpha graphics. Therefore, the alpha value must be equal to 1 if writing and reading of image pixels must be robust, for example for iterative image editing.*

7.3.5 Draw lines

DRAW ARC X,Y, R, A1,A2, D
draws arc with center at point [x,y], radius [r], start angle [a1], stop angle [a2] and direction [d]. If [d] is 0 then arc is clockwise, otherwise it is counterclockwise. Direction parameter can be omitted, in this case arc is clockwise.

DRAW LINE TO X,Y
Draws line from current pen coordinates to point [x,y]

DRAW LINE X1,Y1 TO X2,Y2
draws line from point [x1,y1] to point [x2,y2].

7.3.6 Draw figures

DRAW CIRCLE X,Y SIZE R
draws circle with center at point [x,y] and radius [r].

DRAW CIRCLE X,Y SIZE RX,RY
draws ellipse with center at point [x,y], radius [rx] in "x" direction and radius [ry] in "y" direction.

DRAW CIRCLE X1,Y1 TO X2,Y2
draws ellipse inside rectangle from point [x1,y1] to point [x2,y2].

DRAW POLY X,Y COUNT N START S
draws polygon with arbitrary number of vertices. Variables [x] and [y] are single-dimensional numeric arrays with "x" and "y" coordinates of vertices, [n] is number of vertices and [s] is index of first vertex.
If parameter COUNT is omitted then maximum number of vertices is used.

If parameter START is omitted then vertices start from first index in arrays.
For example:

```
DRAW POLY x,y
```

draws polygon using all vertices in [x] and [y] arrays.
Command OPTION BASE has its effect in this command.

DRAW QUAD X1,Y1, X2,Y2, X3,Y3, X4,Y4
draws quadrangle with vertices at points [x1,y1], [x2,y2], [x3,y3] and [x4,y4].
The points in the parameter list of the QUAD functions must be given in cyclic order. If not, the result is two separate triangles instead of a quadrangle.

DRAW RECT X,Y SIZE R
draws rectangle with center at point [x,y] with size [r] from center to edges in both directions.

DRAW RECT X,Y SIZE RX,RY
draws rectangle with center at point [x,y] with size [rx] from center to edge in "x" direction and [ry] in "y" direction.

DRAW RECT X1,Y1 TO X2,Y2
draws rectangle with corners at [x1,y1] and [x2,y2] coordinates.

DRAW TRI X1,Y1, X2,Y2, X3,Y3
draws triangle with vertices at points [x1,y1], [x2,y2] and [x3,y3].

FILL CIRCLE X,Y SIZE R
fills circle with center at point [x,y] and radius [r].

FILL CIRCLE X,Y SIZE RX,RY
fills ellipse with center at point [x,y], radius [rx] in "x" direction and radius [ry] in "y" direction.

FILL CIRCLE X1,Y1 TO X2,Y2
fills ellipse inside rectangle with corners at [x1,y1] and [x2,y2] coordinates.

FILL POLY X,Y COUNT N START S
fills polygon with arbitrary number of vertices. Variables [x] and [y] are single-dimensional numeric arrays with "x" and "y" coordinates of vertices, [n] is number of vertices to draw and [s] is index of first vertex. If parameter COUNT is omitted then maximum number of vertices is used. If parameter START is omitted then vertices start from first index in arrays. For example:

```
FILL POLY x,y
```

fills polygon using all vertices in [x] and [y] arrays.
Command OPTION BASE has its effect in this command.

FILL QUAD X1,Y1, X2,Y2, X3,Y3, X4,Y4
fills quadrangle with vertices at points [x1,y1], [x2,y2], [x3,y3] and [x4,y4].
The points in the parameter list of the QUAD functions must be given in cyclic order. If not, the result is two separate triangles instead of a quadrangle.

FILL RECT X,Y SIZE R
fills rectangle with center at point [x,y] with size [r] from center to edge in both directions.

FILL RECT X,Y SIZE RX,RY
fills rectangle with center at point [x,y] with size [rx] from center to edge in "x" direction and [ry] in "y" direction.

FILL RECT X1,Y1 TO X2,Y2
fills rectangle with corners at [x1,y1] and [x2,y2] coordinates.

FILL TRI X1,Y1, X2,Y2, X3,Y3
fills triangle with vertices at points [x1,y1], [x2,y2] and [x3,y3].

7.3.7 Images and screenshots

See also **camera** input and output commands in subchapter 2.4 on page 19.

ALBUM EXPORT F$
exports image or video file [f$] from smart BASIC to device camera roll.

ALBUM IMPORT F$
imports image or video file from device camera roll to smart BASIC file [f$].

DRAW IMAGE N$ AT X,Y SCALE S ANGLE A
DRAW IMAGE N$ AT X,Y SCALE SX,SY ANGLE A
draws image with filename [n$] at coordinates [x,y] with scale value [s] on x- and y-axis or [sx] on x-axis and [sy] on y-axis, and rotated clockwise at angle value [a].
Parameter **SCALE** can be omitted to use scale value 1.
Parameter **ANGLE** can be omitted to use angle value 0.
Valid image types are: JPG, JPEG, PNG, BMP, GIF, TIF, ICO, CUR, XBM.
OPTION IMAGE POS command affects this command.
To determine required scale, the size of the image can be obtained with the command GET IMAGE N$ SIZE W,H

DRAW IMAGE N$ IN X1,Y1, X2,Y2, X3,Y3, X4,Y4
draws image with filename [n$] inside arbitrary quadrangle with coordinates: [x1,y1] - top left corner, [x2,y2] - top right corner, [x3,y3] - bottom left corner, [x4,y4] - bottom right corner. Valid image types are: JPG, PNG, BMP, GIF, TIF, ICO, CUR, XBM.
See Examples/Graphics/flying turtle.txt

GET IMAGE N$ DPI_SCALE S
gets dpi-scale of image [n$] to variable [s]. When dpi is 72 the scale is equal to 1.

GET IMAGE N$ SIZE W,H
gets width and height of image [n$] to variables [w] and [h].

GRAPHICS SAVE X,Y, W,H TO N$
Saves to file [n$] the part of graphics screen, which is located at coordinates [x], [y], having width [w] and height [h]. Valid image file types are: JPG, PNG. If file extension is not set then PNG file type is used.

8. Sprites

8.1. General

Sprites are separate graphics layers which do not depend upon main graphics window. They can have custom size and can be moved. Main graphics window is of maximum size and cannot be moved. Sprites are displayed above the background of the main graphics window and have their own position, rotation, transparency and other parameters. Like any object in smart BASIC, each sprite must have its own unique name.
Similarly to interface objects, sprite belongs to the page which was active when sprite was created. Sprite coordinates are relative to the page coordinates. More about pages see in chapter 5“Interface objects” on page 29.

8.1.1 Initial commands

Before using a sprite, it must be created first by one of the following three methods:

- SPRITE BEGIN … SPRITE END, to draw a sprite
- SPRITE LOAD, to load a sprite from file
- SPRITE SCAN, makes sprite from partial screenshot

These commands are explained with examples in subchapter 8.4 on page 51.
It is recommended to use PNG image files to load and save sprite images, because this file format supports image transparency.

8.1.2 Sprite visibility

Sprite visibility on the screen is defined by commands
SPRITE SHOW and SPRITE HIDE.

8.1.3 Animation

Sprites can be single-frame or multi-frame. Besides frame animation available for multi-frame sprites, animation commands like SPRITE PLAY or SPRITE LOOP can perform regular animation of different sprite parameters (coordinates, rotation angle, scale) which can be set with 'SPRITE N$ DA a DS s DX x DY y' command. For example, program:

```
OPTION ANGLE DEGREES
SPRITE "mySprite" DELAY 0.04
SPRITE "mySprite" DA 5
SPRITE "mySprite" LOOP
```

will rotate sprite "mySprite" over 5 degrees every 0.04 seconds. If this is a multi-frame sprite then its frame animation will also be performed.

8.1.4 Sprite display priority

By default, sprites display order depends on order of their creation, but it may be changed using command SPRITE N$ ORDER K. See also "Sprite order rules" on page 55.

8.2. Sprite presets

OPTION SPRITE POS CENTRAL

OPTION SPRITE POS NORMAL
set sprites position mode: "CENTRAL" - sprite position coordinates set center of the sprite; "NORMAL" - sprite position coordinates set top left corner of the sprite. Default is NORMAL.

SPRITE N$ ALPHA X
sets transparency of sprite with name [n$] to value [x]. Valid values are from 0 to 1.

8.3. Get sprite info

GET SPRITE N$ ANGLE X
gets current rotation angle of sprite [n$] to numeric variable [x]. OPTION ANGLE command affects this command.

GET SPRITE N$ DPI_SCALE S
gets dpi-scale of sprite [n$] to numeric variable [s]. When dpi is 72 the scale is equal to 1.

GET SPRITE N$ FRAME X
gets current frame number of sprite [n$] to numeric variable [x]. OPTION BASE command affects this command.

GET SPRITE N$ POS X,Y
gets current coordinates of sprite [n$] to numeric variables [x] and [y].

GET SPRITE N$ SCALE S
GET SPRITE N$ SCALE X,Y
gets current scale of sprite [n$] on both axis to numeric variable [s] or scale on x-axis to variable [x] and on y-axis to variable [y].

GET SPRITE N$ SIZE W,H
gets width and height of sprite [n$] to numeric variables [w] and [h], in points. To get sprite width and height in pixels they should be multiplied to sprite dpi-scale.

SPRITES_COLLIDE (A$, B$)
returns 1 if sprite with name [a$] is currently colliding with sprite [b$], otherwise returns 0.

SPRITE_HIT (N$, X,Y)
returns 1 if point with coordinates [x], [y] hits the sprite [n$]. Otherwise returns 0.

SPRITE_PLAYS (N$)
Returns 1 if sprite with name [n$] is currently animating, otherwise returns 0.

SPRITE_VISIBLE (N$)
returns 1 if sprite with name [n$] is shown, otherwise returns 0.

8.4. Sprite creation, loading, saving and initiation

Sprites can be drawn or loaded from file. Once created they can be modified, copied and/or combined to multi-frame sprites. Creating a sprite by drawing should begin with SPRITE N$ BEGIN W,H. After finishing with SPRITE END the sprite can be modified by starting with SPRITE N$ BEGIN and again finishing with SPRITE END if the modification is done.

SPRITE N$ BEGIN
switches graphics screen to sprite drawing mode in **existing sprite** with name [n$]. Usual graphics commands operate in this mode, but they draw current sprite only instead of drawing in common graphics window. Common graphics window is visible in this mode. Sprite drawing must be finished with command SPRITE END, which switches back to common graphics window, the background graphics layer.

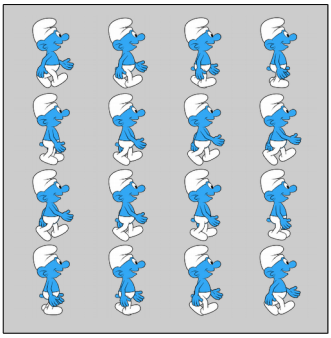
SPRITE N$ BEGIN W,H
switches graphics screen to sprite drawing mode for sprite with name [n$], and sets its size to [w] points in width and [h] points in height. Usual graphics commands operate in this mode, but they draw current sprite only instead of drawing in common graphics window. In this mode common graphics window is not visible. Sprite drawing must be finished with command SPRITE END, which closes sprite graphics window and switches to common graphics view.

SPRITE N$ COPY M$
Copies sprite with name [n$] to new sprite with name [m$]. Screen display order of sprite [m$] is the same as of sprite [n$]. If it is a multi-frame sprite then frames time interval is also copied.

SPRITE N$ LOAD F$, X,Y
Creates multi-frame sprite with name [n$] from the contents of image file (sprite sheet) [f$] which contains [x] sprites horizontally and [y] sprites vertically. Valid image types are: JPG, JPEG, PNG, BMP, GIF, TIF, ICO, CUR, XBM.
Command loads a single sprite in several phases (frames) of motion from one sprite

sheet.
It loads the sprite sheet F$ into sprite N$, divide it into frames (X images in width and Y frames in height) and combine single multi-frame sprite from these images. This multi-frame sprite can be animated, for example it can be a running character, or an exploding bomb, or whatever.
Example of such a sprite sheet is: "sprite_smurf" . Download it from:
http://bit.ly/1DN8gYw
The figure is a reduced version. The grey background indicates the transparency.
The following code shows the smurf in motion:

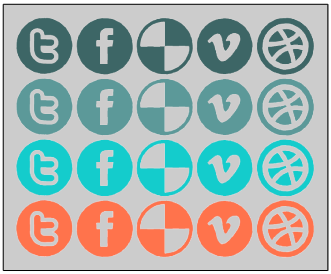
```
GRAPHICS
SPRITE "smurf" LOAD "sprite_smurf.png",4,4
SPRITE "smurf" SHOW
SPRITE "smurf" DELAY 0.04
SPRITE "smurf" LOOP
DO
UNTIL 0
```



It loads the sprite sheet file, creates a multi-frame sprite from this sprite sheet, shows sprite on screen, sets animation speed and runs it in an infinite loop.
Of course you'll need file "sprite_smurf.png" to be present in the current directory. A multi-frame sprite can also be generated from an 'animated GIF' . See command SPRITE N$ LOAD F$

SPRITES M$ LOAD F$, X,Y

Loads image (sprite sheet) with name [f$], creates from it [x] sprites horizontally and [y] sprites vertically, and saves their names to one-dimensional string array [m$]. Command loads a series of sprites from a single sprite sheet F$. Names are stored to array M$.
This approach is used when loading textures, background objects or other images which should be used as sprites. Example of such a sprite sheet is: "sprite_icons".
Download it from: *http://bit.ly/1DN8gYw*
It contains icons in several colors, for twitter, Facebook etc.



The following code is an example program:

```
GRAPHICS
OPTION BASE 1
sw=SCREEN_width()
sh=SCREEN_HEIGHT()
SPRITES M$ LOAD "sprite_icons.png",5,4
dx=sw/6 ! dy=sh/5
' ---- load sprite
FOR i=1 TO 4
  FOR j=1 TO 5
    n=j+(i-1)*5
    SPRITE M$(n)AT (j-1)*dx,(i-1)*dy
    SPRITE M$(n)SHOW
  NEXT j
NEXT i
pause 1
```

```
' ---- position sprites randomly
t=0.5
DO
  IF n>20 THEN n=1
  SPRITE M$(n) AT RND(0.9*sw),RND(0.9*sh)
  n=n+1
  PAUSE t
  t=t-t/30
UNTIL 0
END
```

It loads the sprite sheet file, creates 20 separate sprites from this sprite sheet, shows them on screen and repositions them randomly in an infinite loop.
Of course you'll need file "sprite_icons.png" to be present in the current directory.

SPRITE N$ DELETE
deletes sprite with name [n$].

SPRITES DELETE
completely deletes all sprites, both visible and hidden.

SPRITE END
SPRITE N$ END
finishes drawing of sprite and switches drawing mode to common graphics screen. This command must be executed to finish sprite creation, started by command SPRITE BEGIN.

SPRITE N$ FLIP K
mirrors sprite [n$] horizontally if [k] = 1, vertically if [k] = 2, or shows sprite without mirroring if [k] = 0.

SPRITE N$ HIDE
hides sprite with name [n$] off the screen.

SPRITE N$ LOAD F$
creates sprite with name [n$] from the contents of image file [f$]. Valid image types are: JPG, JPEG, PNG, BMP, GIF, TIF, ICO, CUR, XBM.
It is recommended to use PNG image files to load and save sprite images, because this file format supports image transparency.
If sprite is loaded from file then you should not delete the file while sprite is in use. This is due to image caching peculiarities in iOS.
If the file F$ is an 'animated GIF' then this load-command will generate a multiframe sprite which can be saved as sprite-sheet.. The following example generates a sprite-sheet from an animated GIF of an explosion. By changing the variable name$, it can be used for other GIF's.

```
' Make SpriteSheet from animated GIF
name$="explosion"
sw=SCREEN_WIDTH()
sh=SCREEN_HEIGHT()
GRAPHICS
GRAPHICS CLEAR
SPRITES DELETE ' delete orphan sprites
IF FILE_EXISTS (name$&".png") THEN GOTO display
SPRITE 1 LOAD name$&".gif"
SPRITE 1 SAVE name$&".png"
SPRITES DELETE
debug pause ' check format x,y of generated spritesheet
```

```
display:
OPTION SPRITE POS CENTRAL
' ---- enter here x,y from generated spritesheet
SPRITE "a" LOAD name$&".png",6,5
SPRITE "a" AT sw/2,sh/2 SCALE 1/2
SPRITE "a" SHOW
SPRITE "a" DELAY 0.04
SPRITE "a" LOOP
DO
UNTIL 0
```

In the first run the sprite-sheet is generated and the program stops at the command 'debug pause'.
The generated sprite-sheet should then be viewed and the format x,y (6,5 in this case) should be set in the command SPRITE "a" LOAD name$&".png",x,y
In the second run the multi-frame sprite is then shown in animation.

SPRITE N$ ORDER K
sets screen depth display order for sprite with name [n$]. By default order indices start with 0. Sprite with higher order index is displayed above the sprite with lower order index. OPTION BASE command affects this command.
See "Sprite order rules" on page 55.

SPRITE N$ RENAME M$
Renames sprite with name [n$] to new name [m$].

SPRITE N$ RESIZE W,H
changes physical size of sprite with name [n$] to [w] points wide and [h] points high.

SPRITE N$ SAVE F$
saves sprite with name [n$] to image file [f$]. Valid image file types are: JPG, JPEG, PNG. If file extension is not set then PNG file type is used.

SPRITE N$ SCAN X,Y, W,H
creates sprite with name [n$] from the part of common graphics screen, which is located at coordinates [x], [y], having width [w] and height [h].

SPRITE N$ SHOW
shows sprite with name [n$] on the screen.

SPRITE N$ STAMP
leaves a stamp of sprite with name [n$] on the common graphics screen using all current parameters of the sprite. It is not necessary for sprite to be visible on the screen when using this command. Command GRAPHICS MODE has its effect in this command.

8.5. Positioning and moving sprites

SPRITE N$ AT X,Y SCALE S ANGLE A FLIP K
SPRITE N$ AT X,Y SCALE SX,SY ANGLE A FLIP K
places sprite with name [n$] at coordinates [x] and [y], sets its scale to value [s] on x- and y-axis or [sx] to x-axis and [sy] to y-axis, and rotates it to angle [a]. Parameter FLIP allows to mirror sprite horizontally if [k] = 1, mirror vertically if [k] = 2, or show without mirroring if [k] = 0 or if FLIP is not used. Scaling and rotation are performed relatively to the center of the sprite. Parameters SCALE, ANGLE and FLIP are optional, their order is arbitrary. By default scale is 1 and angle is 0. OPTION ANGLE command affects this command.

SPRITE N$ DA a DS s DX x DY y
SPRITE N$ DA A DS SX,SY DX X DY Y
sets regular change of specified parameters of the sprite [n$] with each frame cycle. DA parameter sets change of angle by value [a], DS - scale by value [s] simultaneously on x- and y-axis or by value [sx] on x-axis and by value [sy] on y-axis, DX - x-coordinate by value [x], DY - y-coordinate by value [y]. Each parameter is optional but at least one should be specified. OPTION ANGLE command affects the value of DA parameter.

SPRITE N$ FLIP K
mirrors sprite [n$] horizontally if [k] = 1, vertically if [k] = 2, or shows sprite without mirroring if [k] = 0.
Example of repeated FLIP usage:

```
Sprite$="submarine 1.png"
sh=SCREEN_HEIGHT()
sw=SCREEN_WIDTH()
movex=3
GRAPHICS
OPTION SPRITE POS CENTRAL
SPRITE "a" LOAD Sprite$
SPRITE "a" DELAY 0.04
SPRITE "a" AT sw/2,sh/2
SPRITE "a" dX movex
SPRITE "a" SHOW
SPRITE "a" loop
DO
PAUSE 1
CALL ReverseSprite
GET TOUCH nn AS x,y
UNTIL x>-1
SPRITES DELETE
END

DEF ReverseSprite ' Function to alternate flip
Flipped=(Flipped+1)%2
SPRITE "a" FLIP Flipped
SPRITE "a" dX .movex*(1-Flipped*2)
END DEF
```

SPRITE N$ PLAY
Starts animation for multi-frame sprite [n$] and starts motion, also for single frame sprites, determined by the parameters set by the command SPRITE N$ DA a DS s DX x DY y. . Animation of multi-frame sprites cycles once and then stops.

8.6. Sprite order rules

by 'matt7' See: *https://bit.ly/2SHAt2V*
1. Sprite with a higher order index is displayed above sprite with a lower order index.

```
Sprite "a" has order 0
Sprite "b" has order 1 -> "b" appears above "a"
```

2. Default sprite order depends on order of creation (order index starts at 0).

```
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
```

3. Sprites can be assigned an order index manually using command **SPRITE N$ ORDER K**.

```
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Sprite "a" order 3 -> "a" has order 3
```

4. Sprites can be assigned to an order index that other sprites also have, in which case the reordered sprite appears above all other sprites of that order index.

```
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Sprite "a" order 1 -> "a" has order 1 ("a" appears above
"b")
```

5. If multiple sprites have the same order index, a sprite can jump to the top of its order index group if one of the following events occur:

a. It is resized

```
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Sprite "a" order 1 -> "a" has order 1 ("a" appears above
"b")
Sprite "b" resized ("b" appears above "a")
```

b. It is made visible after previously being hidden

```
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Sprite "a" order 1 -> "a" has order 1 ("a" appears above "b")
Sprite "b" hidden
Sprite "b" shown ("b" appears above "a")
```

c. It is copied (the copy jumps to the top)

```
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Sprite "a" order 1 -> "a" has order 1 ("a" appears above "b")
Sprite "b" copy to "b2" -> "b2" has order 1 ("b2" appears
above "a")
```

6. If a sprite is copied, the copy will have the same order index as the original (see "b" above)

7. If the last created sprite is deleted, the next created sprite will reuse the last order number.

```
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Sprite "c" deleted
Sprite "d" created -> "d" has order 2
```

Note: Sprite "d" has order 3 if "c" is not deleted, and "d" stays at order 3 if "c" is deleted after "d" is created

8. If a sprite created before the last sprite is deleted, the next created sprite will continue with the next order number.

```
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Sprite "b" deleted
Sprite "d" created -> "d" has order 3
```

9. If a sprite is reordered to the order index for the next sprite that will be created, the next sprite will still use that order index.

```
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Sprite "a" order 3 -> "a" has order 3
Sprite "d" created -> "d" has order 3
```

10. Default sprite order index for next sprite to be created is maintained across pages.

```
Page "P1" set
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Page "P2" set
Sprite "d" created -> "d" has order 3
Sprite "e" created -> "e" has order 4
Sprite "f" created -> "f" has order 5
```

11. Sprite order cannot be manually assigned if the page it belongs to is not set as the active page.

```
Page "P1" set
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Page "P2" set
Sprite "a" order set to 3 -> "a" has order 0 (no effect)
```

12. Sprite will jump to the currently active page if it is shown after previously being hidden

```
Page "P1" set
Sprite "a" created -> "a" has order 0
Sprite "b" created -> "b" has order 1
Sprite "c" created -> "c" has order 2
Page "P2" set
Sprite "a" shown (no effect)
Sprite "a" hidden
Sprite "a" shown -> "a" now appears on page "P2" (with the
same coordinates but now relative to page "P2")
```

#11 was my main problem, as I had no clue why some of my sprite order commands were working (I didn't realize this was only on the active page) while other sprite order commands seemed to be doing nothing (on pages that were visible but weren't the active page).

#12 was also a surprise and turned out to also be giving me problems, as it helped explain why some of my sprites weren't reappearing with a sprite show command. They were being moved to another page that happened to be hidden! (In my case, this was an active page that was hidden because it contains all my touch controls/touch targets that I wanted to be invisible.)

8.7. Multi-frame sprites

Purpose of multi-frame sprites is to show animation as a single sprite.
To simplify the creation of such sprites, the required separate phases of motion are stored as frames in a single image: a sprite sheet.
Sprite sheets are commonly used in games and storybooks and are widespread on internet. The first step, after finding the desired image-file, is to load the sprite sheet

and to separate the frames with the command SPRITE N$ LOAD F$, X,Y or to separate the sheet into single sprites with the command SPRITES M$ LOAD F$, X,Y. Note the essential difference: SPRITE or SPRITES.

SPRITE N$ ADD K$
Adds contents of sprite with name [k$] to contents of sprite with name [n$], thus creating multi-frame sprite.

SPRITE N$ COPY M$
Copies sprite with name [n$] to new sprite with name [m$]. Screen display order of sprite [m$] is the same as of sprite [n$]. If it is a multi-frame sprite then frames time interval is also copied.

SPRITE N$ DELAY X
Sets to value [x] the time interval between frames when animating multi-frame sprite [n$], in seconds. By default it is equal to 0.1.

SPRITE N$ FRAME K
sets current frame number to [k] for multi-frame sprite [n$]. OPTION BASE command affects this command.

SPRITE N$ LOOP
starts animation for multi-frame sprite [n$]. Animation loops infinitely.
Also the motion parameters will be activated as determined by the command SPRITE N$ DA a DS s DX x DY y
Animation can be stopped by the command SPRITE N$ PLAY. In the following example the smurf stops moving after one second plus one 'play'-cycle:

```
GRAPHICS
SPRITE "smurf" LOAD "sprite_smurf.png",4,4
SPRITE "smurf" SHOW
SPRITE "smurf" DELAY 0.04
SPRITE "smurf" LOOP
PAUSE 1
SPRITE "smurf" PLAY
DO ! UNTIL 0
```

SPRITE N$ PLAY
Starts animation for multi-frame sprite [n$]. Animation cycles once and then stops. Furthermore this command starts motion, also for single frame sprites, determined by the parameters set by the command SPRITE N$ DA a DS s DX x DY y

SPRITE_PLAYS (N$)
Returns 1 if sprite with name [n$] is currently animating, otherwise returns 0.

SPRITE N$ SWAY
starts animation for multi-frame sprite [n$]. When the end of animation cycle is reached, it starts in the opposite direction of cycle. Animation loops infinitely.

SPRITE_FRAME (N$)
returns current frame number for sprite [n$]. OPTION BASE affects this function.

9. Music, sound and speech

Smart BASIC can playback audio files, play musical notes and generate speech.

- Playing audio files, e.g. sound and music, in formats *.WAV, *.MP3 and *.AIFF is performed with MUSIC commands, such as MUSIC LOAD or MUSIC PLAY.
- Playing musical notes is performed according to the MIDI protocol with NOTES commands and can be done in real time or from loaded musical composition.
- Speech is generated from text by synthesized voices.

Furthermore a simple attention-signal is generated with the command:

BEEP
makes short beep sound.

9.1. Playing audio files

The files containing data referred to as 'musical composition' should contain audio data coded in AIFF-, MP3- or WAV-format.

MUSIC M$ DELETE
deletes musical composition [m$] and releases its resources.

MUSIC M$ LOAD F$
loads musical composition [m$] from file [f$] in such formats as *.WAV, *.MP3, *.AIFF.

MUSIC M$ LOOP
starts playback of musical composition [m$], looping it infinitely.

MUSIC M$ PLAY
starts playback of musical composition [m$].

MUSIC M$ STOP
stops playback of musical composition [m$].

MUSIC M$ TIME T
sets playback point of musical composition [m$] at [t] seconds.

MUSIC_LENGTH (M$)
returns duration of musical composition [m$], in seconds.

MUSIC_PLAYING (M$)
returns 1 if musical composition [m$] is currently playing. Otherwise returns 0.

MUSIC_TIME (M$)
returns current playback point of musical composition [m$], in seconds.

9.2. Playing musical notes and MIDI compositions

Playing musical notes can be done (a) in real time or (b) from loaded musical composition:
(a)Playing notes in real time is performed with NOTES MIDI command. So, your device can be used as a MIDI synthesizer. A detailed description of MIDI commands is out of scope of this manual. A short survey is given in the table next to the description of the NOTES MIDI command
(b)Playing notes from loaded musical composition can be done in several steps:

- loading musical composition from file with NOTES LOAD command or from musical notation with NOTES SET command.
- start/stop playing of musical composition with NOTES PLAY/NOTES STOP commands. Musical playing does not block program execution and is performed in

background while program is running.
After finishing musical composition, playing is not stopped automatically. To detect end of playing the functions NOTES_TIME() and NOTES_LENGTH() are used.
Total 16 musical tracks are available for playing notes. According to MIDI standard, 10-th track is used for percussions only.
Default musical instruments can be used (see "Instruments" section on page 62) or they can be loaded from files in Soundfont2 or DLS formats with NOTES INSTRUMENTS command.

NOTES INSTRUMENTS F$
sets bank of instruments in Soundfont2 or DLS format from file [f$].

NOTES INSTRUMENTS DEFAULT
sets default bank of instruments.

NOTES LOAD F$
loads musical composition in *.MID format from file [f$].

NOTES MIDI T, CMD,A,B
sends MIDI command to device, where:
[t] - track number (0..15)
[cmd] - MIDI command,
[a], [b] - command parameters.
The table gives a survey of the MIDI commands.
For example, commands:

```
NOTES MIDI 3,12,19
NOTES MIDI 3,9,60,127
```

set instrument number 19 for track 3 and start playing C4 note with volume 127 (“velocity” in MIDI-terms) on 3-rd track.

MIDI commands		
nr	**command**	**parameters**
8	note-off	key #; release velocity
9	note-on	key #; attack velocity
10	aftertouch	key #; key pressure
11	control-change	controller #; controller data
12	patch-change	instrument #
13	channel-pressure	channel pressure
14	pitch-bend	lsb; msb
15	system-message	none or sysex

More info on MIDI commands can be found on the forum, e.g. at
http://bit.ly/1N4mR2e
Special info on the drum sounds can be found on
http://bit.ly/1JHiSOn

Note numbers

12-C0	24-C1	36-C2	48-C3	60-C4	72-C5	84-C6	96-C7	108-C8
13-C#0	25-C#1	37-C#2	49-C#3	61-C#4	73-C#5	85-C#6	97-C#7	109-C#8
14-D0	26-D1	38-D2	50-D3	62-D4	74-D5	86-D6	98-D7	110-D8
15-D#0	27-D#1	39-D#2	51-D#3	63-D#4	75-D#5	87-D#6	99-D#7	111-D#8
16-E0	28-E1	40-E2	52-E3	64-E4	76-E5	88-E6	100-E7	112-E8
17-F0	29-F1	41-F2	53-F3	65-F4	77-F5	89-F6	101-F7	113-F8
18-F#0	30-F#1	42-F#2	54-F#3	66-F#4	78-F#5	90-F#6	102-F#7	114-F#8
19-G0	31-G1	43-G2	55-G3	67-G4	79-G5	91-G6	103-G7	115-G8
20-G#0	32-G#1	44-G#2	56-G#3	68-G#4	80-G#5	92-G#6	104-G#7	116-G#8
21-A0	33-A1	45-A2	57-A3	69-A4	81-A5	93-A6	105-A7	117-A8
22-A#0	34-A#1	46-A#2	58-A#3	70-A#4	82-A#5	94-A#6	106-A#7	118-A#8
23-B0	35-B1	47-B2	59-B3	71-B4	83-B5	95-B6	107-B7	119-B8

Note numbers for percussions (Track № 10)

35 Bass Drum	51 Ride Cymbal	67 High Agogo
36 Kick Drum	52 China Cymbal	68 Low Agogo
37 Snare Cross Stick	53 Ride Bell	69 Cabasa
38 Snare Drum	54 Tambourine	70 Maracas
39 Hand Clap	55 Splash cymbal	71 Whistle Short
40 Electric Snare Drum	56 Cowbell	72 Whistle Long
41 Floor Tom 2	57 Crash Cymbal 2	73 Guiro Short
42 Hi-Hat Closed	58 Vibraslap	74 Guiro Long
43 Floor Tom 1	59 Ride Cymbal 2	75 Claves
44 Hi-Hat Foot	60 High Bongo	76 High Woodblock
45 Low Tom	61 Low Bongo	77 Low Woodblock
46 Hi-Hat Open	62 Conga Dead Stroke	78 Cuica High
47 Low-Mid Tom	63 Conga	79 Cuica Low
48 High-Mid Tom	64 Tumba	80 Triangle Mute
49 Crash Cymbal	65 High Timbale	81 Triangle Open
50 High Tom	66 Low Timbale	82 Shaker

NOTES PLAY

starts/resumes playing of musical composition, loaded with NOTES LOAD or NOTES SET command. Avoid starting new musical composition while previous musical composition is still playing.

NOTES SAVE F$

saves musical composition to *.MID file [f$].

NOTES SET A$,B$,...

sets musical composition from musical notation. Each specified string value [a$], [b$], ... is a separate musical track, written in musical notation according to the rules:

- letters "C", "D", "E", "F", "G", "A", "B" are notes "do", "re", "mi", "fa", "sol", "la", "si"
- letter "R" is a rest
- characters "#" and "$" make sharp and flat notes, e.g.: C#, D$
- octave is a number after note, e.g. E3, D#5
- letters "W", "H", "Q", "I", "S", "T" are duration of following notes, where "W"=whole, "H"=half, "Q"=quarter, "I"=eighth, "S"=sixteenth, "T"=thirty second, e.g. HC#
- character "." means that next note is one and a half times longer: Q.C
- letter "V" with integer number (0..127) set volume of following notes, e.g.: V60C#
- character ":" preceding with integer number set musical instrument number, e.g. 123:C
- musical instrument bank number (if present in bank of instruments) can be specified before instrument number, separated with character "/":, e.g. 2/12:E
- chord is set with notes in round brackets, e.g. (EGB)
- octave number, note duration, volume, instrument number are not necessary to specify for each and every note - they affect all following notes until the value is changed

Up to 16 musical tracks can be used. Track number 10 is for percussion instruments. In percussions each note means separate instrument. List of instruments is given in "Instruments" section.

By default, 4-th octave, one fourth note duration, volume 127, instrument number 0 are used.
Examples:

```
NOTES SET "c c# d d# e f f# g g# a a# b"
NOTES SET "12:c5cggaahg qffeeddhc"
NOTES SET "(egb)ccc (fac5)d4dd","19:we2 f"
NOTES SET ,,,,,,,,,"c2cc icqc icqc c"
```

NOTES STOP
stops playing of musical composition.

NOTES TEMPO N
sets tempo to [n] for already loaded musical composition or for newly created with command NOTES SET musical compositions. Default tempo is 120.

NOTES_LENGTH ()
returns length of musical composition.

NOTES_PLAYING ()
returns 1 if musical composition is playing now, otherwise returns 0.

NOTES_TIME ()
returns current playback time.

Example with quarter notes and chords

Example from rbytes:

```
NOTES SET "46:q(c6e6g6)(d6f6a6)"
NOTES PLAY
```

The q sets duration as quarter notes. The notes inside parentheses play together as a chord, 46 is just one of 127 MIDI instruments you can use, it is the Orchestral Harp.

9.3. Musical instruments

Default musical instruments:

Piano:	**Chromatic Percussion:**	**Organ:**
0 Acoustic Grand Piano	8 Celesta	16 Drawbar Organ
1 Bright Acoustic Piano	9 Glockenspiel	17 Percussive Organ
2 Electric Grand Piano	10 Music Box	18 Rock Organ
3 Honky-tonk Piano	11 Vibraphone	19 Church Organ
4 Electric Piano 1	12 Marimba	20 Reed Organ
5 Electric Piano 2	13 Xylophone	21 Accordion
6 Harpsichord	14 Tubular Bells	22 Harmonica
7 Clavinet	15 Dulcimer	23 Tango Accordion

Guitar:	**Bass:**	**Strings 1:**
24 Acoustic Guitar (nylon)	32 Acoustic Bass	40 Violin
25 Acoustic Guitar (steel)	33 Electric Bass (finger)	41 Viola
26 Electric Guitar (jazz)	34 Electric Bass (pick)	42 Cello
27 Electric Guitar (clean)	35 Fretless Bass	43 Contrabass
28 Electr. Guitar (muted)	36 Slap Bass 1	44 Tremolo Strings
29 Overdriven Guitar	37 Slap Bass 2	45 Pizzicato Strings
30 Distortion Guitar	38 Synth Bass 1	46 Orchestral Harp
31 Guitar harmonics	39 Synth Bass 2	47 Timpani

Strings 2:
48 String Ensemble 1
49 String Ensemble 2
50 Synth Strings 1
51 Synth Strings 2
52 Choir Aahs
53 Voice Oohs
54 Synth Voice
55 Orchestra Hit

Brass:
56 Trumpet
57 Trombone
58 Tuba
59 Muted Trumpet
60 French Horn
61 Brass Section
62 Synth Brass 1
63 Synth Brass 2

Reed:
64 Soprano Sax
65 Alto Sax
66 Tenor Sax
67 Baritone Sax
68 Oboe
69 English Horn
70 Bassoon
71 Clarinet

Pipe:
72 Piccolo
73 Flute
74 Recorder
75 Pan Flute
76 Blown Bottle
77 Shakuhachi
78 Whistle
79 Ocarina

Synth Lead:
80 Lead 1 (square)
81 Lead 2 (sawtooth)
82 Lead 3 (calliope)
83 Lead 4 (chiff)
84 Lead 5 (charang)
85 Lead 6 (voice)
86 Lead 7 (fifths)
87 Lead 8 (bass + lead)

Synth Pad:
88 Pad 1 (new age)
89 Pad 2 (warm)
90 Pad 3 (polysynth)
91 Pad 4 (choir)
92 Pad 5 (bowed)
93 Pad 6 (metallic)
94 Pad 7 (halo)
95 Pad 8 (sweep)

Synth Effects:
96 FX 1 (rain)
97 FX 2 (soundtrack)
98 FX 3 (crystal)
99 FX 4 (atmosphere)
100 FX 5 (brightness)
101 FX 6 (goblins)
102 FX 7 (echoes)
103 FX 8 (sci-fi)

Ethnic:
104 Sitar
105 Banjo
106 Shamisen
107 Koto
108 Kalimba
109 Bag pipe
110 Fiddle
111 Shanai

Percussive:
112 Tinkle Bell
113 Agogo
114 Steel Drums
115 Woodblock
116 Taiko Drum
117 Melodic Tom
118 Synth Drum

Sound effects:
119 Reverse Cymbal
120 Guitar Fret Noise
121 Breath Noise
122 Seashore
123 Bird Tweet
124 Telephone Ring
125 Helicopter
126 Applause
127 Gunshot

9.4. Speech

LIST VOICES
prints list of voices to the screen.

LIST VOICES TO A$,N
saves list of voices for SAY command to string array [a$] and size of returned array to numeric variable [n].

SAY CONTINUE
continues text pronunciation, paused by SAY PAUSE command.

SAY PAUSE
pauses text pronunciation, started by SAY TEXT command.

SAY PITCH X
sets voice pitch for command SAY TEXT to the value [x].
Valid values are from 0 to 2. By default 1.

SAY RATE X
sets speech rate for command SAY TEXT to the value [x]. Valid values are from 0 to 2. By default 1.

SAY STOP
stops text pronunciation, started by SAY TEXT command.

SAY TEXT T$
says text [t$] with voice. Requires iOS 7 and newer.

SAY VOICE V$
sets voice for SAY TEXT command. List of available voices you can get with LIST VOICES command. Empty string "" sets native voice.

SAY VOLUME X
sets volume for command SAY TEXT to the value [x]. Valid values are from 0 to 1. By default 1.

SAYING ()
returns 1 if speech is currently pronounced by SAY TEXT command, otherwise returns 0.

10. Networking

To display contents of a webpage, a browser should be created on the current page. See subchapter 5.9“Browsers” on page 36 in “Interface objects“

About HTTP communication

If error prevents HTTP command from execution, then error message can be received by HTTP_ERROR$() function. If server returns response after execution of HTTP command, then this response can be received by HTTP_RESPONSE$() function. This can be useful when executing HTTP POST and HTTP POSTDIM commands, which send data to server and have no input parameters.
When executing HTTP commands it is optionally possible to set values which are present in HTTP request header. For this purpose one-dimensional string array in HEADER parameter of HTTP commands is used, elements of this array should be strings formatted as "header_field : value", for example:

```
s$ = "This is my message to server"
h$(1) = "content-type:text/html"
h$(2) = "content-length:"&len(s$)
HTTP "posttestserver.com/post.php" HEADER h$ POST s$
PRINT HTTP_RESPONSE$()
```

Simple web browser

On the forum a simple web browser has been posted by 'Dave', As it is realized in SmartBasic, it can be adapted to your own needs. “Simple web browser with bookmarks (iPad/iPhone/iPod touch)” can be found on *http://bit.ly/1PpeSOS*

HTTP URL$ HEADER H$ GET T$

performs HTTP GET request to address [url$] and stores server response to string variable [t$]. HEADER parameter defines contents of HTTP request header and is optional, but if used then [h$] must be one-dimensional string array; see its format details in preface.
Example:

```
HTTP "kibernetik.pro" GET t$
PRINT t$
```

HTTP URL$ HEADER H$ GETDIM M, N

performs HTTP GET request to address [url$] and stores server response to one-dimensional numeric array [m].
Size of array [m] is changed according to amount of data received and is stored to numeric variable [n], which is optional.
HEADER parameter defines contents of HTTP request header andis optional, but if used then [h$] must be one-dimensional string array; see details in preface.
Example:

```
HTTP "google.com/images/logo.png" GETDIM m
FILE "google.png" WRITEDIM m
```

HTTP URL$ HEADER H$ HEAD T$

performs HTTP HEAD request to address [url$] and stores server response to string variable [t$].
HEADER parameter defines contents of HTTP request header and is optional, but if used then [h$] must be one-dimensional string array; see its format details in preface.

Example:
```
HTTP "apple.com" HEAD t$
PRINT t$
```

HTTP URL$ HEADER H$ POST T$
performs HTTP POST request with contents of [t$] to address [url$]. HEADER parameter defines contents of HTTP request header and is optional, but if used then [h$] must be one-dimensional string array; see its format details in preface.

HTTP URL$ HEADER H$ POSTDIM M, N
performs HTTP POST request to address [url$], contents of request is taken from one-dimensional numeric array [m] as [n] number of bytes.
Array [m] must contain only values from 0 to 255.
If bytes to send [n] is omitted then all contents of array [m] will be sent. HEADER parameter defines contents of HTTP request header and is optional, but if used then [h$] must be one-dimensional string array. See its format details in preface.
Example:
```
FILE "image.jpg" READDIM m,n
h$(1) = "content-type:image/jpeg"
h$(2) = "content-length:" & n
HTTP url$ HEADER h$ POSTDIM m
```

HTTP_ERROR$ ()
returns error message if error occurred when executing last HTTP command. If command was executed without errors then empty string "" is returned.

HTTP_HEADER$ ()
returns HTTP header of last HTTP command response.

HTTP_RESPONSE$ ()
returns server response on last HTTP command.
If server responded nothing then empty string "" is returned.
Example:
```
HTTP "posttestserver.com" POST "Help me!"
PRINT HTTP_RESPONSE$()
```

HTTP_STATUS ()
returns HTTP status code of last HTTP command response.
See RFC 2616 for HTTP status code details.
Note: “RFC 2616” is superseded and split into several documents. See info on *https://www.mnot.net/blog/2014/06/07/rfc2616_is_dead*

PING (H$)
performs ping of host [h$] and returns 1 if host is available or 0 if not.

PING (H$,P)
performs ping of port [p] of host [h$] and returns 1 if port is available or 0 if not.

SYSTEM_EXT_IP$ ()
returns external IP address of device. If unavailable then returns empty string "".

SYSTEM_INT_IP$ ()
returns local IP address of device. If unavailable then returns empty string "".

11. Miscellaneous

11.1. Date and Time functions

CURRENT_YEAR () returns current year

CURRENT_MONTH () returns current month. (january=1, …)

CURRENT_DATE () returns current day of month

CURRENT_DAY () returns current day of the week (sunday=0, …)

CURRENT_HOUR () returns current hour

CURRENT_MINUTE () returns current minute

CURRENT_SECOND () returns current second

PAUSE X
pauses program for [x] seconds.

TIME ()
returns time since program start or since time reset by TIME RESET command, in seconds.

TIME RESET
sets to 0 time, returned by TIME () function.

11.2. Program launch and discontinuing

DEBUG PAUSE
pauses program execution and displays debug screen with contents of all variables at this moment.

DEBUG PAUSE X
performs DEBUG PAUSE command, but at first delays for [x] seconds.

END
ends program execution.

EXIT
ends program execution and quits application.

LAUNCHER$ ()
returns string indicating how the program was launched: "appstore" if as a stand-alone application; "basic" if from smart BASIC application; "desktop" if by desktop icon.

PAUSE X
pauses program for [x] seconds.

RUN N$
runs program with name [n$]. It is the same as ending your current program, loading file with name [n$] and running it. You can omit ".txt" extension in file name.

SLOWDOWN
makes CPU idle for a short time period, thus reducing power consumption.
This command can be used in waiting cycles, for example when waiting for button press, and also in looping places where execution speed is not as important as low power consumption.
For example, this program uses CPU at 100% while performing simple looping:

```
1 GOTO 1
```

and this program leaves CPU almost idle although looping much slower because of short CPU pause in SLOWDOWN command:

```
1 SLOWDOWN
GOTO 1
```

STOP
terminates program execution. This is the same as terminating program with (X) button.

11.3. GPS

To receive GPS data you need to turn on GPS with command SET GPS ON.
GPS data are accumulating as they arrive, and GPS_COUNT() function returns number of accumulated GPS locations. Every GPS location contains latitude, longitude, altitude, speed, course, horizontal and vertical accuracies of current point.
GPS data can be read with GET GPS command.
Example programs can be found in the folder '/Examples/Hardware'
Note for earliest iPad-users (from: http://ipad.about.com):

> *"The Wi-Fi model of the iPad does not have an Assisted-GPS chip, but it can locate the user by using Wi-Fi triangulation. This isn't quite as accurate as Assisted-GPS, but it is fairly accurate."*

Without built-in GPS you will need some kind of connection (cellular or wifi) to get location information.

COMPASS_ACCURACY ()
returns current compass accuracy (if compass is present), measured in degrees. If accuracy is undefined or if interference is strong returns -1.

COMPASS_HEADING ()
returns current direction relative to true north, obtained from electronic compass (if present). Heading is defined by top side of program interface and is measured in degrees, from 0 to 360. If direction is undefined then returns -1.

GET GPS LAT X LON Y ALT Z SPD S CRS C HAC H VAC V
saves GPS latitude to [x], longitude to [y], altitude to [z], speed to [s], course to [c], horizontal accuracy to [h] and vertical accuracy to [v].
Latitude, longitude and course are measured in degrees, height and accuracy - in meters, speed - in meters per second.
If parameter is currently undefined then returns -1.
Accuracy value -1 means that respective data are invalid. Any parameter is optional, but at least one must be used. Parameters order is arbitrary.
Examples:

```
GET GPS ALT altitude
GET GPS LAT x LON y ALT z
GET GPS COURSE m(0) SPEED m(1)
```

GPS_COUNT ()
returns number of pending GPS positions.

GET GPS DIRECTION H
gets GPS data of current direction and saves it to numeric variable [h]. Direction is measured in degrees.

SET GPS OFF
SET GPS ON
turns on and turns off GPS. ON = enabled, OFF = disabled.

Notes on GPS data

About GPS data access
GPS supplies its data not when you need them but when they are available. And GPS is not waiting for your program to receive its data - either you read them right now or you miss them.
That is why smart BASIC buffers all GPS data and allows you to read them with your own speed without fear that you missed something. If you don't need complete route but only latest value then you can skip all data and use only latest one.
GPS supply its data very fast (many times per second), and they all are stacked in memory until you read them. So, asking for new data only once a minute means accessing only very old GPS readings. This means you need to call 'GET GPS' until 'GPS_COUNT()' is zero, otherwise you accumulate unread GPS data.
See usage in 'GPS Track': *http://kibernetik.pro/forum/viewtopic.php?f=20&t=1491*

About speed and distance
The distance traveled in a brief interval during a trip can be calculated, of course, as the difference between the coordinates of the start and end. At small distances however, so at low speeds and short intervals, that method gives inaccurate results.
In that case and actually in general, the lap-distance can better be calculated by multiplying the speed with the time interval.
GPS receivers typically calculate velocity by measuring the frequency shift (Doppler shift) of the GPS carrier-signal in three dimensions. Velocity accuracy can be scenario dependent, (multi-path, obstructed sky view from the dash of a car, mountains, city canyons, …) but 0.2 m/sec per axis is achievable. This accuracy of 20cm is not possible by calculating the distance between two GPS-coordinates.

About global distances
The distance between locations is determined by the angle between the coordinates and the distance to the reference-point. The GPS-system reference-level for the distance to the reference point is the so-called WGS84 mean earth radius which is determined at 6371.008666 km.
See info at Wikipedia: *https://en.wikipedia.org/wiki/World_Geodetic_System*
The 'radius' at your position is increased with the height above sea-level as in the following code. The formula for distance is from Wikipedia:
https://en.wikipedia.org/wiki/Great-circle_distance

```
OPTION ANGLE DEGREES
TwoPI= 6.28318531
'
'--- Function, 'Lat'=latitude, 'Long'=longitude
DEF Distance(lat1,lon1,lat2,lon2,alt)
'alt is average: (alt1+alt2)/2
dLong=ABS(lon1-lon2)
term1=SIN(Lat1)*SIN(Lat2)
term2=COS(Lat1)*COS(Lat2)*COS(dLong)
Angle=ACOS(term1+term2)
dist=Angle/360*.TwoPi*(6371008.666+alt)
RETURN dist
END DEF
```

11.4. Device properties

ACCEL_X ()

ACCEL_Y ()

ACCEL_Z ()
return device acceleration values on "x", "y" and "z" axes. Accelerometer "x" and "y" axes are identical to screen axes, and "z" axis is perpendicular to screen. Accelerometer also responds to device rotation.

BATTERY_LEVEL ()
returns current battery charge level, from 0 to 100 percent.

BATTERY_STATE ()
returns current battery charging state: 0 is discharging, 1 is charging.

DEVICE_TYPE$ ()
returns device type. Possible values: "iPad", "iPhone".

DEVICE_NAME$ ()
returns device name.

GET ORIENTATION P
saves current interface orientation to numeric variable [p].
1=portrait orientation
2=landscape orientation with "Home" button on the left
3=portrait orientation upside down
4=landscape orientation with "Home" button on the right

KEYBOARD_VISIBLE ()
returns 1 if screen keyboard is currently visible. Otherwise returns 0. This function works only when the keyboard is solid and is at the bottom of the screen (external keyboard connected via Bluetooth).

LAUNCHER$ ()
returns string indicating how the program was launched: "appstore" if as a stand-alone application; "basic" if from smart BASIC application; "desktop" if by desktop icon.

OPTION SCREENLOCK OFF
OPTION SCREENLOCK ON
turn off/on idle device screen locking when program is running.
If your device is left idle for a specified amount time, its screen locks and turns off. You can disable this idle screen locking with OPTION SCREENLOCK OFF/ON command to prevent your device from switching off when your program is running.

SET BRIGHTNESS X
sets screen brightness to the value [x]. Possible values of [x] are from 0 to 1.

SET ORIENTATION ALL
SET ORIENTATION 0
unlocks orientation.

SET ORIENTATION TOP
SET ORIENTATION BOTTOM
SET ORIENTATION LEFT
SET ORIENTATION RIGHT
SET ORIENTATION PORTRAIT

SET ORIENTATION VERTICAL
SET ORIENTATION LANDSCAPE
SET ORIENTATION HORIZONTAL
sets device interface orientation.

SET ORIENTATION P
sets device interface orientation according to value [p].
[p]=0: unlocks orientation
[p]=1: sets portrait orientation
]p]=2: landscape orientation with "Home" button on the left
[p]=3: portrait orientation upside down
[p]=4: landscape orientation with "Home" button on the right
[p]=5: portrait orientation with any direction
[p]=6: landscape orientation with any direction

SET TOOLBAR BATTERY OFF

SET TOOLBAR BATTERY ON
turns OFF and turns ON display of battery charge level on the top control toolbar.

SET TOOLBAR TIME OFF

SET TOOLBAR TIME ON
turns OFF and turns ON display of current time on the top control toolbar.

SLOWDOWN
makes CPU idle for a short time period, thus reducing power consumption.
This command can be used in waiting cycles, for example when waiting for button press, and also in looping places where execution speed is not as important as low power consumption. For example, this program uses CPU at 100% while performing simple looping:

```
1 GOTO 1
```

and this program leaves CPU almost idle although looping much slower because of short CPU pause in SLOWDOWN command:

```
1 SLOWDOWN
GOTO 1
```

SYSTEM_LANGUAGE$ ()
returns current user language in format ISO 639-1.

SYSTEM_VERSION()
returns iOS version number.

SYSTEM_VOLUME ()
returns current device volume level, from 0 to 1.

11.5. Available fonts

LIST FONTS
Prints list of available fonts to the screen.

LIST FONTS TO A$,N
Saves list of available fonts to string array [a$] and size of returned array to numeric variable [n].

12. User interface settings

12.1. Text encoding/decoding

Smart BASIC allows you to encode text of the program. Encoded program can be run, but its text is not available for any user.
To encode program you need to:

- Input coding password using OPTION CODEPASS command.
- Rename program by changing file's extension from .TXT to .COD

Program file with .COD extension is a coded program. It can be run or included inside another program, but its text cannot be viewed or edited.
Coded program can be decoded only if the same password is set which was used when encoding the program.
To decode already encoded program you need to:

- Input the same coding password which was used when encoding the program using OPTION CODEPASS command.
- Rename program by changing file's extension from .COD to .TXT

There is no need to input coding password each time when encoding or decoding. Coding password can be entered only once, and it will be saved.

Note from the forum

Code:

```
OPTION CODEPASS "password"
```

where "password" is your password.
*This should be done **NOT IN YOUR CODED** program. This should be done in a separate program - this is just a one-time setting for your device.*
*And this is enough. This should be done **ONLY ONCE**. You will not need to do it ever again: your device will remember it.*
After you set your password, you will be able to encode ANY your programs with this password.
You may ask: why you need to set a password at all?
Answer: it will be possible to decode encoded program ONLY if the device where program should be decoded is set to the same password as you set before.

OPTION CODEPASS P$
enters password [p$] for program text encoding/decoding. Password is saved inside the device.

12.2. UNDO/REDO *when editing code*

On iPhone shake your device to get dialog to undo editing.
On iPad UNDO / REDO buttons are present on additional keyboard layout.

12.3. Code marking

You can set tinted background in code editor for any lines of your code. To do this you need to set special color marks in the beginning of code line. Color marks are: 'r', 'g', 'b', 'c', 'm', 'y' for red, green, blue, cyan, magenta and yellow tint colors respectively. If you need to stop colouring at some line, use '' (two ' characters) color mark.

Example of tinted code:

```
'r'
PRINT A 'line with red tint
'g'
PRINT B1 'lines with green tint
PRINT B2
''
PRINT C 'line without tint
'b'
PRINT D 'line with blue tint
```

Tinting is applied to background of any color. Tinting is available for iOS 5.0 and higher.

12.4. App preferences and presets

The following commands should be executed to set global preferences or to perform other single-time operations. These settings are permanent until reset.

OPTION DROPBOX OFF

OPTION DROPBOX ON
disables/enables Dropbox integration. See subchapter 6.6 on page 41.

RESTORE EXAMPLES
restores deleted or modified example files.

SET EDITOR CAPSYNTAX OFF
SET EDITOR CAPSYNTAX ON
turns ON or turns OFF automatic smart BASIC language keywords highlighting with capital letters in code editor during text editing. This setting changes the text of program, not only its visual representation. This effect cannot be undone.

SET EDITOR DEFAULT
sets editor font settings to default.

SET EDITOR BACK COLOR R,G,B
sets editor background color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

SET EDITOR FONT COLOR R,G,B
sets editor font color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

SET EDITOR FONT NAME N$
sets name of editor font to [n$]. List of fonts you can get by command LIST FONTS.

SET EDITOR FONT SIZE N
sets size of editor font to value [n].

Fit font-size to screen-width
with monospaced fonts, e.g. “Courier” or "Menlo-Regular", the number of characters per line can be determined with the following code:

```
GET SCREEN SIZE sw,sh
GET SCREEN SIZE sw,sh
fmin=10 'minimum fontsize
fsize=MAX((sw-20)/(0.6*n),fmin)
SET EDITOR FONT SIZE fsize
```

SET UNDERGROUND OFF

SET UNDERGROUND ON
turns OFF or turns ON the main, most power-consumptive calculation cycle of application when application is sent to background. By default underground is off, i.e. application is NOT active when it is in the background.

12.5. Custom skins

After SET UI commands, which set user interface settings, SET UI APPLY command should be used, which makes settings to be applied.
When setting images, RETINA versions of these images can be used together with main images. RETINA version of image has double resolution and "@2x" characters at the end of file name.
For example, when using command:

```
SET UI RUN ICON "image.png"
```

"image@2x.png" image can be used together with "image.png" image. In this case if device screen is standard then "image.png" is used, if screen is RETINA then "image@2x.png" image is automatically used.

SET UI APPLY
applies user interface settings.

SET UI CLOUD_IN ICON N$
sets icon to enter cloud storage from file [n$].

SET UI CLOUD_OUT ICON N$
sets icon to exit from cloud storage from file [n$].

SET UI COPY ICON N$
sets icon for copy files operation from file [n$].

SET UI CUT ICON N$
sets icon for cut files operation from file [n$].

SET UI DEBUG BACK N$
sets debug window background from file [n$].

SET UI DEBUG FONT_COLOR R,G,B
sets debug table font color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

SET UI DEBUG FONT_NAME N$
sets name of debug table font to [n$].

SET UI DEBUG FONT_SIZE N
sets size of debug table font to value [n].

SET UI DEBUG TABLE_COLOR R,G,B
sets debug table color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

SET UI DEFAULT
sets user interface settings to default.

SET UI DELETE ICON N$
sets icon for delete files operation from file [n$].

SET UI EDIT ICON N$
sets icon to edit program text from file [n$].

SET UI ENTER ICON N$
sets icon to go next operation from file [n$].

SET UI EXIT ICON N$
sets icon to return back operation from file [n$].

SET UI FILE_BAR BACK N$
sets file bar background from file [n$].

SET UI FILE_BAR FONT_COLOR R,G,B
sets file bar icon titles font color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

SET UI FILE_BAR FONT_FILE F$
sets font file [f$], if file bar icon titles use third-party TTF-font.

SET UI FILE_BAR FONT_NAME N$
sets name of file bar icon titles font to [n$].

SET UI FILE_BAR FONT_SIZE N
sets size of file bar icon titles font to value [n].

SET UI FILES BACK N$
sets files list window background from file [n$].

SET UI FILES COPY_BACK N$
sets selected file to copy background from file [n$].

SET UI FILES CUT_BACK N$
sets selected file to cut background from file [n$].

SET UI FILES FONT_COLOR R,G,B
sets files list font color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

SET UI FILES FONT_NAME N$
sets name of files list font to [n$].

SET UI FILES FONT_FILE F$
sets font file [f$], if files list uses third-party TTF-font.

SET UI FILES FONT_SIZE N
sets size of files list font to value [n].

SET UI FILES ICON N$
sets icon to display files list from file [n$].

SET UI FILES SELECT_BACK N$
sets selected file background from file [n$].

SET UI FILES SELECT_COLOR R,G,B
sets selected file font color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

SET UI FORUM ICON N$
sets icon to visit Support Forum from file [n$].

SET UI GRAPHICS BACK N$
sets graphics window background from file [n$].

SET UI HELP ICON N$
sets icon to show help from file [n$].

SET UI NEW_FILE ICON N$
sets icon to create new program file from file [n$].

SET UI NEW_FOLDER ICON N$
sets icon for create new folder operation from file [n$].

SET UI PASTE ICON N$
sets icon for paste files operation from file [n$].

SET UI PAUSE ICON N$
sets icon to pause program execution from file [n$].

SET UI PREVIEW ICON N$
sets icon to preview image file from file [n$].

SET UI RENAME ICON N$
sets icon for rename file operation from file [n$].

SET UI RUN ICON N$
sets icon to run program from file [n$].

SET UI SEARCH ICON N$
sets icon to search text from file [n$].

SET UI STOP ICON N$
sets icon to stop program execution from file [n$].

SET UI TOOL_BAR BACK N$
sets tool bar background from file [n$].

SET UI TOOL_BAR FONT_COLOR R,G,B
sets tool bar font color to value with red [r], green [g] and blue [b] components. Valid values are from 0 to 1.

SET UI TOOL_BAR FONT_FILE F$
sets font file [f$], if toolbar uses third-party TTF-font.

SET UI TOOL_BAR FONT_NAME N$
sets name of tool bar font to [n$].

SET UI TOOL_BAR FONT_SIZE N
sets size of tool bar font to value [n].

SET UI TOOL_BAR ICONS_COLOR R,G,B
sets tool bar icons color to value with red [r], green [g] and blue [b] components.

13. Notes

• *Using libraries*

You can insert contents of another program file in your program by using "{}" brackets. For example, code:

```
{library.txt}
```

will insert text from file "library.txt" at this place in your program. You can omit extension ".txt" and if you write it like this:

```
{library}
```

then smart BASIC will understand that you want the contents of file "library.txt" here. If file should be inserted only if it was not inserted yet, file name should be inclosed in double brackets:

```
{{library}}
```

This is a pre-processing feature, so at first smart BASIC combines all files into one

code and only then it executes the program.
If error occurs then smart BASIC loads that file which contained the error to display.
It is suggested for you to create "lib" folder in the root of your file system and to keep all your code libraries there. Then you can access your library from any file in any folder like this:

```
{/lib/functions}
{/lib/const}
```

Slash character '/' before "lib" indicates that this is a root folder. It will make your library to be path-independent from your program files. Also it will be compatible if you share your programs and libraries with others.

- ***Compacting code lines***

You can write multiple commands in one line by separating them with "!" symbol.
For example:

```
A=B+1 ! C=B*2
```

Commands starting with "END" like END IF, END DEF and so on, can be written in one word as ENDIF, ENDDEF.

- ***Code autosave***

Code is automatically saved when you run the program or when exit editing mode.

- ***Duplicating your code***

You can duplicate program by copying the file and pasting it back to the same folder. A new file is then created by adding word "copy" to the file name. If that file already existed then it is deleted and replaced with fresh copy of file.
For example, if you duplicate file "program.txt" then its copy with name "program copy.txt" will be created. If file "program copy.txt" already existed then it will be replaced with fresh copy.

- ***Hiding keyboard***

You can hide keyboard when editing your code by pressing program title on the top toolbar.

- ***Running program with desktop icon***

Steps to create desktop icon:

- Open blank page in Safari browser.
- In address-line write link to the program which will be launched with this icon.
  Link is written as address "*http://kibernetik.pro/sb.php*?" and then path to the program.
  Space character can be substituted with "%20" characters.
  File extension ".txt" and root folder "/" can be omitted. Other extensions, e.g. “.sb” should be given.
  Example: to run "hello world.txt" program from root folder the address will be: http://kibernetik.pro/sb.php?hello%20world
- Go to this address in Safari. Safari launches the program which is indicated in the link.

Customised desktop launcher

- Return back to Safari to the page which run the program and perform standard export of this link to the desktop. It will then ask for a name which is “Favourites” as default. You can change that. After completion the icon will occur on the desktop. The figure shows an example with custom name (“Dutchman”) and default image.

Coupling with launcher
If you generate a desktop-icon which runs “/- Launcher.sb”, see 'Personal notes' on page 6, then you get an easy and fast way to access your favourite programs.

> If you **start the program in Basic**, **then it will return in Basic**.
> If you **start the program in Launcher**, either via desktop or in Basic,
> **then the program will return to Launcher**
> **If the Launcher is stopped, then it returns to where it was started:**
> on the desktop ('Home'-page) or in Smart BASIC

Customise icon-image
It is possible to provide the desktop-icon with custom image. Indicate then in the address string after symbol "&" the internet address of the image.
For example:
&*http://icons.iconarchive.com/icons/treetog/i/128/Set-Program-Default-icon.png*
It is however a problem to find a website where the available Icon-images are available for usage in this way.

- ***Your program in App Store***

You can share your creation with millions of people!
See instructions on the forum: *http://kibernetik.pro/forum/viewforum.php?f=34*

- ***Support forum***

Visit forum (*http://kibernetik.pro/forum*) to share and discuss programming topics in English and Russian.

14. Examples

The folder 'Examples' contains several folders with example-programs. These programs can be modified in order to get experience with the functions and commands.

RESTORE EXAMPLES
restores deleted or modified example files.

Content of 'Examples':

Complex numbers
- fractal in colors
- fractal
- simple complex

Editor Settings
- black and white
- blue night
- default
- fairy
- gold day
- gold night
- iOS 5 fonts list
- night vision
- typewriter

Games
- 15.txt
- Fractaler in colors
- Fractaler
- MAZER
- life
- *Folders:*
  - In Cell

Graphics
- circles
- color lines
- color pixels
- flying turtle
- interactive turtle
- pollution
- round fun
- sierpinski
- winter
- *Folders:*
  - images

Hardware
- GPS
- accelerometer
- my GPS position

Interactive Interface
- bar graphs
- browsers
- multitouch
- pages
- painter 2
- painter
- palette demo pie
- palette parameters
- pseudo depth
- sin + cos
- speech
- *Images:*
  - flower.jpg
- *Folders:*
  - Motion Detector

Music & Sound
- notes.txt
- play MIDI file
- sfx test 2
- sfx test
- synthesizer
- *Folders:*
  - files

Simple
- heavy math

Skins
- blue skin
- default
- *Folders:*
  - files

Smart Features
- I am a hardcore coder
- code highlighting
- using libraries
- *Folders:*
  - lib

Sprites
- collide me
- every frame
- fall 'n' crash
- screen intro
- sprite creator
- sprite delay
- sprites position
- *Images:*
  - knight6.png
  - test10.png

Appendix A. Obsolete commands

The following commands are obsolete and replaced by new commands. Obsolete commands will work as they always worked, but they are just not documented anymore.

• *Obsolete in version 2.5 and later*

- DEBUG ON/OFF, replaced with: DEBUG PAUSE

• *Obsolete in version 3.0 and later*

- GET GPS DIRECTION, replaced with: COMPASS_HEADING ()
- GPS_POSITION_AVAILABLE() and
- GPS_DIRECTION_AVAILABLE()
  replaced with: GPS_COUNT ()
- GET GPS POSITION
  replaced with: GET GPS LAT x LON y ALT z SPD s CRS c HAC h VAC v

• *Obsolete in version 4.0 and later*

- GRAPHICS LOCK / UNLOCK, replaced with: REFRESH
- IOS_VERSION (), replaced with: SYSTEM_VERSION()

• *Obsolete in version 4.2 and later*

- SLIDER N$ VALUE K AT X,Y HSIZE S
- SLIDER N$ VALUE K AT X,Y VSIZE S
  Use SLIDER N$ VALUE K AT X,Y SIZE S ANGLE A instead

• *Obsolete in version 4.6 and later*

- BUTTON SET TITLE replaced with: BUTTON SET TEXT
- BUTTON TITLE replaced with: BUTTON TEXT
- TIMER RESET replaced with: TIME RESET
- TIMER() replaced with: TIME()

• *Obsolete in version 4.8 and later*

and are replaced with	=> new equivalents:
- BROWSER SET TEXT	=> BROWSER TEXT
- BROWSER SET URL	=> BROWSER URL
- BUTTON SET TEXT	=> BUTTON TEXT
- FIELD SET TEXT	=> FIELD TEXT
- LIST SET SELECTION	=> LIST SELECT
- LIST SET TEXT	=> LIST TEXT
- SLIDER SET VALUE	=> SLIDER VALUE
- SWITCH SET STATE	=> SWITCH STATE