OliBasic Reference

OliBasic

Written by aFox

OliBasic version 3.00

Originally published as De_Re_Basic!

Original Author Paul Laughton, 2011

Edited by Robert A. Rioja

robrioja@gmail.com

http://www.RvAdList.com

Document Version 2024-08-02

Table of Contents

I - INTRODUCTION

1 OliBasic

OliBasic is an Android interpreter for the Basic language, based on ***RFO-BASIC!***, which is often referred to as ***BASIC!*** For purposes of generality, this document will often refer to ***OliBasic*** as ***BASIC!***. Please note that ***Basic*** refers to the general definition of the language, while ***BASIC!*** refers to the specific Basic implementations ***RFO-BASIC!*** and ***OliBasic***.

2 Credits

Thanks to **Paul Laughton**, the original creator of RFO-BASIC! and its original documentation. The first edition was published in 2011. Mr. Laughton placed that document in the Public Domain in 2016.

Thanks also to **Mike Leavitt** for his many contributions and long-time support.

Thanks to long time RFO-BASIC! collaborator and forum contributor **Mougino** (**Nicolas Mougin)**, developer of the GW library and the Basic! compiler, who also developed the original cover art. Also for his code contributions to OliBasic.

Thanks to forum contributor **Spike** for relentless support and great documentation.

Thanks to forum contributor **aFox** (**Gregor**) for implementing OliBasic. Without aFox there would be no OliBasic.

Thanks to forum contributor **humpty** who graciously donated much of his hBasic code to OliBasic. Some of his donations are in the areas of GPS, Word_all$, Debug.dump.fn. Notify.status, Device$(), Globals.fnimp, Graphics, etc.

Thanks to forum contributor **Tinine** (**Craig**) for his support and proofreading abilities.

Thanks to forum contributor **tino1003870** (**Tino**) for his support and his BLE contributions.

And, of course, thanks to **George Boole**, who taught us the value of 0 and 1.

3 Disclaimer

OliBasic and all documentation are provided with no warranty. Although the authors will make an effort to ensure correctness, the software and documentation are provided “as is”, with any faults, defects, bugs, and errors.

4 Documentation

This document, ***OliBasic Reference***, was developed from the original ***De_Re_BASIC!*** document and aFox's original OliBasic documentation. It is a companion to the ***OliBasic! User Manual***, which was also developed from the original ***De_Re_BASIC!*** document.

This ***OliBasic Reference*** and the ***OliBasic User Manual*** were edited, and are maintained, by Robert A. Rioja.

The latest version of this document can be found at https://www.RvAdList.com.

This ***OliBasic Reference*** contains the following sections:

5 References

OliBasic website:	https://olibasic.gitlab.io/About/index.html
BASIC! forum:	https://www.tapatalk.com/groups/rfobasic/
Spike's RFO-Basic! manual:	https://rfobasic.miraheze.org/
Mougino's website:	http://mougino.free.fr/
My website:	https://www.RvAdList.com

II - SYNTAX

1 Syntax and Command Description

1.1 A Program

A program is made up of lines of text. With a few exceptions that will be explained later, each line of text is one or more **statements**. If a line has more than one statement they are separated by colon (":") characters.

A statement always consists of a single command, usually followed by one or more parameters that are separated by commas. Here is a simple program:

```
Print "Hello, World!"
```

This program has one statement. The command is **`Print`**. It has one parameter, the string constant **"Hello, World!"**. A string constant, or string literal, is a set of characters enclosed in double quotation marks. This, too, will be explained later.

1.2 Multiple Commands on a Line

More than one source code statement may be written on one physical line. Statements are separated by the colon character ":". For example, the following line uses three separate statements to initialize some variables:

```
name$ = "BASIC!" : ver=1.86 : Array.load reviews$[], "Great!", "Wow!"
```

There are two commands, **`Sensors.open`** and **`Sql.update`**, which use the colon as a sub-parameter separator. If you use multiple-command lines, be careful when using these two commands.

1.3 Line Continuation

A source code statement may be written on more than one physical line using the line continuation character "~". If "~" is the last thing on a line, except for optional spaces, tabs, or a '%' comment, the line will be merged with the next line. This behavior is slightly different in the **`Array.load`** and **`List.add`** commands; see the descriptions of those commands for details.

Note: this operation is implemented by a preprocessor that merges the source code lines with continuation characters before the source code is executed. If you have a syntax error in the merged line, it will show as one line in the error message, but it will still be multiple lines in the editor. Only the first physical line will be highlighted, regardless of which line the error is in.

For example, the code line:

```
s$ = "The quick brown fox " + verb$ + " over " + count$ + " lazy dogs"
```

could be written as:

```
s$ = "The quick brown fox " + ~
verb$ + ~ % what the fox did
" over " + ~
count$ + ~ % how many lazy dogs
" lazy dogs"
```

1.4 Comments

1.4.1 ! - Single Line Comment

If the first character in a line is the "!" character, the entire line is considered a comment and is ignored. If the "!" appears elsewhere in the line, it does not indicate a comment.

1.4.2 Rem - Single Line Comment (legacy)

If the first three characters of a line are "Rem", "REM", or even "rEm", the entire line is considered a comment and is ignored. If any of these combinations appear elsewhere in the line, it does not indicate a comment.

1.4.3 !!, /* and */ - Block Comment

When a line begins with the "!!" characters, all lines that follow are considered comments and are ignored. The Block Comment section ends at the next line that starts with "!!". The same can be achieved by starting the block with "/*" and ending it with "*/".

Example:

```
...
!! start of comment
...
!! end of comment
...
```

Example:

```
...
/* start of another comment
...
*/ end of second comment
...
```

Note that the entire line is considered a comment, including all text before and after the "!!" or "*/".

1.4.4 %, // - Middle of Line Comment

If the "%" character, or the "//" characters appear anywhere in a line (except within a quoted string) then rest of the line is a comment.

Example:

```
...
% this is a comment
Print "this % is not a comment"   % this is a comment
...
```

Example:

```
...
// this is a comment
Print "this // is not a comment"   // this is a comment
...
```

1.5 Upper and Lower Case

Command names are described using both upper and lower case for ease of reading. For the sake of

consistency as well as ease of reading, this document follows these rules:

1. If the name is just one word, only the first letter is capitalized. For example: **`Print`** or **`Pause`** or **`Else`**.
2. If the name is two or more concatenated words, the first letter of each word is capitalized. For example: **`GoTo`** or **`WakeLock`** or **`ElseIf`**.
3. If the name is multi-word with underscores or periods separating the words, only the first letter of the first word is capitalized. For example: **`Audio.start`** or **`Array.binary.search`** or **`Is_infinite`**.
4. If the name is a combination of rules 2 and 3, then both rules apply. For example: **`KeyDown.on`** or **`ConsoleTouch.resume`**.
5. If the name is a combination of rules 2 and 3, but any part after an underscore or period is concatenated, then the first letter of the concatenated part s not capitalized. For example: **`Bt.onReadReady.resume`** or **`Adoc.lastModified`**.

Unfortunately, there are illogical exceptions to the above rules, mostly for historical reasons.

When the program is run, every character (except those between double quotation marks) is assumed to be lower case by the interpreter. Therefore, the above rules are only to make this manual easier for humans to read. When you write your programs, you can use any capitalization rules you like, since commands like **`Else`**, **`ELSE`**, **`else`** and **`eLse`** are all interpreted the same way.

1.6 <nexp>, <sexp> and <lexp>

These notations denote a numeric expression (<nexp>), a string expression (<sexp>), and a logical expression (<lexp>). An expression can be a variable, a number, a quoted string or a full expression such as (a*x^2 + bx + c).

1.7 <nvar>, <svar> and <lvar>

This notation is used when a variable, not an expression, must be used in the command. Arrays with indices (such as n[1, 2] or s$[3, 4]) are considered to be the same as <nvar>, <svar> and <lvar>.

1.8 Array[] and Array$[]

This notation implies that an array name without indices must be used.

1.9 Array[{<start>, <length>}] and Array$[{<start>, <length>}]

In most contexts, numeric expressions inside the brackets are indices specifying a single array element. In some commands, a pair of numeric expressions specifies a segment of the array. Both the start index and length are numeric expressions, and both are optional. This notation is shorthand for:

```
Array [ { {<start_nexp>} {, <length_nexp>} } ]
Array$ [ { {<start_nexp>} {, <length_nexp>} } ]
```

1.10 {something}

Indicates something optional.

1.11 { A | B |C }

This notation indicates that a choice of either A, B, or C, must be made. For example:

```
Text.open {r|w|a}, fn…
```

Indicates that either "r" or "w" or "a" must be chosen:

```
Text.open r, fn…
Text.open w, fn…
Text.open a, fn…
```

1.12 X, …

Indicates a variable-sized list of items separated by commas. At least one item is required.

1.13 {,n} …

Indicates an optional list with zero or more items separated by commas.

1.14 <statement>

Indicates an executable statement. A <statement> is usually a line of code but may occur within other commands such as: **`If`** <lexp> **`Then`** <statement>.

1.15 Optional Parameters

Many statements have optional parameters. If an optional parameter is omitted, the statement assumes a default value or performs a default action.

If an optional parameter is omitted, use a comma to mark its place, so following parameters are handled correctly. However, if there are no following parameters, omit the comma, too. With a few special exceptions (like Print), no statement can end with a comma.

2 Numbers

For the purposes of this documentation, numbers that appear in a program are called Numeric Constants.

2.1 Decimal Numbers

Decimal numbers can be typed in a program:

- A leading sign ("+" or "-"), a decimal point ("." only), and an exponent (power of 10) are optional.
- An exponent is "e" or "E" followed by a number. The number may have a sign but no decimal point.

If you use a decimal point, it MUST follow a digit. So **0.15** is a valid number, but **.15** is a syntax error.

Normally, numbers are stored as double-precision floating point (64-bit IEEE 754) type. This means:

- A printed number will always have decimal point. For example, 99 will print as "99.0". You can print numbers without decimal points by using the **Int$()** or **Format$()** functions. For example, either **Int$(99)** or **Format$("##", 99)** will print "99".
- A number with more than 7 significant digits will be printed in floating point format. For example, the number 12345678 will be printed as 1.2345678E7. **Int$()** or **Format$()** can be used to print large numbers in other than floating point format.
- Mathematical operations on decimal values are imprecise. If you are working with currency you should multiply the number by 100 until you need to print it out. When you print it, divide by 100.

A logical value (false = 0, true <> 0) is a kind of number.

You can use string functions to convert numbers to strings. **Str$()**, **Int$()**, **Hex$()** and a few others do simple conversions. **Format$()** and **Using$()** can do more complex formatting.

2.2 BigDecimal Numbers

For more precise calculations, the BigDecimal type is available.

A BigDecimal is an exact way of representing numbers. A Double has a certain precision. Working with doubles of various magnitudes (say d1=1000.0 and d2=0.001) could result in the 0.001 being lost when adding because the difference in magnitude is so large. BigDecimal avoids this problem because Double is Base 2, whereas BigDecimal is Base10.

The disadvantage of BigDecimal is that it's slower, and it's a bit more difficult to program algorithms. This is because variables are strings that represent numbers, and math operations are replaced by commands. For example:

```
B = 20
A = B + 3
```

becomes

```
BigD.fromDouble B$, 20
BigD.add A$, B$, "3"
```

If you are dealing with money, or if precision is a must, use BigDecimal. Otherwise Doubles tend to be good enough.

Here is an example using standard Double math:

```
A = 0.02
```

```
B = 0.03
C = B - A
Print C
```

This will print 0.009999999999999998 instead of the expected 0.01 value.

Now here is the same example using BigDecimal math:

```
BigD.fromDouble A$, 0.02
BigD.fromDouble B$, 0.03
BigD.subtract C$, B$, A$
Print C$
```

This will print 0.01, as expected.

2.3 NaN (Not a Number) or Infinity values (IEEE 754)

These are floating point values in our case from type Double.
If you try SQR (-1) or 0/0 you get a NaN, because it is an undefined operation.
But if you try 1/0 you get an Infinity, because it is an infinite result.
Or you get these values per definition like VAL("Infinity"), VAL("-Infinity") and VAL("NaN").

This differs from older Basic implementations and in some parts also from RFO-Basic 1.91.
That should not have a negative impact on older code. Hopefully in this development stage, you will only get a runtime error if you want to handle these values and functions, where input values must be of the type Integer or Long.
Functions which use BigDecimal inside do not deal with NaN (Not a Number) or Infinity values.
Examples: BigD(decimal) command group, List.join
Functions which use <lexp> and <nexp> interpret these values as 0.
For detection use Is_NaN and Is_Infinite. The last one returns -1 if the value is negative.

!NaN Example

```
Dim x[1]
x[1] = 1
Array.std_dev nanV, x[ ] % Returns NaN in all Basic! versions.
If nanV Then ? "true" : Else ? "false"  % → true   But that is wrong, see below
If Is_Number("NaN") Then ? "true" : Else ? "false"  % → true
If Is_NaN(nanV) Then ? "true" : Else ? "false"  % → true
If nanV = nanV Then ? "true" : Else ? "false"  % → false
If nanV <> nanV Then ? "true" : Else ? "false"  % → true
If nanV < 0 Then ? "true" : Else ? "false"  % → false
If nanV > 0 Then ? "true" : Else ? "false"  % → false
```

!Infinity Example

```
!infiV = -1/0             % Returns -Infinity.
infiV = Val("-Infinity")  % Returns -Infinity in all Basic! versions.
? infiV
If infiV Then ? "true" : Else ? "false"  % → true
If Is_Number("-Infinity") Then ? "true" : Else ? "false"  % → true
If Is_Infinite(infiV) Then ? "true" : Else ? "false"  % → true
?  Is_Infinite(infiV)                    % → -1
If infiV = infiV Then ? "true" : Else ? "false"  % → true
If infiV <> infiV Then ? "true" : Else ? "false"  % → false
If infiV < 0 Then ? "true" : Else ? "false"  % → true
If infiV > 0 Then ? "true" : Else ? "false"  % → false
```

Use instead of:

```
If nanV Then ? "true" : Else ? "false"  % → true   But that is wrong
```

```
If !Is_NaN(nanV) & nanV Then ? "true" : Else ? "false"  % → false   That is
right
While nanV % → true   But that is wrong
While !Is_NaN(nanV) & nanV  % → false   That is right
Until nanV % → true   But that is wrong
Until !Is_NaN(nanV) & nanV  % → false   That is right
```

Sum of Arrays

```
nanV = Val("NaN")
? nanV
If Is_NAN(nanV) Then ? "true" : Else ? "false"  % → true
infiV = Val("-Infinity") % Returns -Infinity in all Basic! versions.
? infiV
If infiV Then ? "true" : Else ? "false"  % → true

Array.load exampl[], 1, 2, 3
Array.sum s, exampl[]
Print s  % → 6

Array.load exampl[], 1, 2, 3, nanV
Array.sum s, exampl[]
Print s  % → NaN

Array.load exampl[], 1, 2, 3, infiV
Array.sum s, exampl[]
Print s  % → -Infinity

Array.load exampl[], 1, 2, 3, nanV, infiV
Array.sum s, exampl[]
Print s  % → NaN
```

Comparison with NaN

A comparison with a NaN always returns an *unordered result* even when comparing with itself. The comparison predicates are either signaling or non-signaling on quiet NaN operands; the signaling versions signal the invalid operation exception for such comparisons. The equality and inequality predicates are non-signaling so $x = x$ returning false can be used to test if x is a quiet NaN. The other standard comparison predicates are all signaling if they receive a NaN operand, the standard also provides non-signaling versions of these other predicates. The predicate *isNaN(x)* determines if a value is a NaN and never signals an exception, even if x is a signaling NaN.

Comparison between NaN and any floating-point value *x* (including NaN and ±∞)						
Comparison	**NaN ≥ *x***	**NaN ≤ *x***	**NaN > *x***	**NaN < *x***	**NaN = *x***	**NaN ≠ *x***
Result	Always **False**	Always **False**	Always **False**	Always **False**	Always **False**	Always **True**

See also
https://docs.oracle.com/cd/E19957-01/806-3568/ncg_intro.html#110
https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

3 Strings

Strings are written as any set of characters enclosed in quote (**"**) characters. The quote characters are not part of the string. For example, **"This is a string"** is a string of 16 characters.

To include the quote character in a string, you must escape it with a backslash: **\"**. For example:

```
Print "His name is \"Jimbo\" Jim Giudice."
```

prints: His name is "Jimbo" Jim Giudice.

Newline characters may be inserted into a string with the escape sequence **\n**:

```
Print "Jim\nGiudice"
```

prints:

```
Jim
Giudice
```

"\n" can mean different things on different systems. In BASIC!, it is the same as an ASCII LF (line feed) character.

You can use another escape sequence, **\t**, to put a TAB character into a string. To embed a backslash, escape it with another backslash: ****. Other special characters can be inserted using the **CHR$()** function.

Strings with numerical characters can be converted to numbers using the VAL(<sexp>) function.

For the purposes of this documentation, strings that appear within a program are called String Constants.

4 Variables

A BASIC! variable is a container for some numeric or string value.

4.1 Variable Names

Variable names must start with the characters "a" through "z", "#", "@", or "_". The remaining characters in the variable name may also include the digits, "0" through "9".

A variable name may be as long as needed.

Upper case characters can be used in variable names but they will be converted to lower case characters when the program is run. The variable name "gLoP" is the same as the name "glop".

You should avoid using variable names that start with command keywords. Such variables are valid under most conditions, as will be explained later in this manual, but their use may cause confusion or errors. For example, `Donut = 5` is interpreted as `Do Nut = 5`. BASIC! thus expects this `Do` statement to be followed by an `Until` statement somewhere before the program ends.

Statement labels and the names of user-defined functions follow the same naming rules as variables.

4.2 Variable Types

There are two types of variables: Variables that hold numbers and variables that hold strings. Variables that hold strings end with the character "$". Variables that hold numbers do not end in "$".

Age, Amount and Height are examples of numeric variable names.

First_Name$, Street$ and A$ are examples of string variable names.

If you use a numeric variable without assigning it a value, it has the value 0.0. If you use a string variable without assigning it a value, its value is the empty string, "".

4.3 Scalar and Array Variables

There are two classes of variables: Scalars and Arrays.

4.4 Scalars

A scalar is a variable that can hold only one value. When a scalar is created it is assigned a default value. Numeric scalars are initialized to 0.0. String scalars are initialized to an empty, zero-length string, "".

You create a scalar variable just by using its name. You do not need to predeclare scalars.

4.5 Arrays

An array is variable that can hold many values organized in a systematically arranged way. The simplest array is the linear array. It can be thought of as a list of values. The array A[index] is a linear array. It can hold values that can be accessed as A[1], A[2],...,A[n]. The number (variable or constant) inside the square brackets is called the index.

If you wanted to keep a list of ten animals, you could use an array called Animals$[] that can be accessed with an index of 1 to 10. For example: `Animals$[5] = "Cat"`.

Arrays can have more than one index or dimension. An array with two dimensions can be thought of as a list of lists. Let's assume that we wanted to assign a list of three traits to every animal in the list of animals. Such a list for a "Cat" might be "Purrs", "Has four legs" and "Has Tail". We could set up the

Traits array to have two dimensions such that Traits$[5, 2] = "Has four legs". If someone asked what are the traits of cat, search Animals$[index] until "Cat" is found at index=5. Index=5 can then be used to access Traits[index,[{1|2|3}].

Arrays can have any number of dimensions of any size.

Arrays are "one-based". This means that the first element of an array has an index of "1". Attempting to access an array with an index of "0" (or less than 0) will generate a run-time error.

Before an array can be used, it must be dimensioned using the **Dim** command. The **Dim** command establishes how many indices are going to be used and the sizes of the indices. Some commands automatically create a one-dimensional array. Auto-dimensioned array details will be seen in the description of those commands.

Note: It is recommended that the List commands (see below) be used in place of one-dimensional arrays. The List commands provide more versatility than the Array commands.

The following commands:

```
Bundle.put mBundle, "mArray", A[]
Bundle.get mBundle, "mArray", B[]
```

will copy array A to array B, losing all references.

4.6 Array Segments

Some commands take an array as an input parameter. If the array is specified with nothing in the brackets (for example, "Animals$[]"), then the command reads the entire array.

Most of these commands allow you to limit their operation to a segment of the array, using the notation "Array[start, length]", where both "start" and "length" are numeric expressions.

For example, you can write "Animals$[2,3]". Usually that means "the animal at row 2 and column 3 of a two dimensional array called Animals$". When used to specify an array segment, it has a different meaning: "read only the segment of the Animals$ array that starts at index 2 and includes 3 items". Notice that this notation applies only to one-dimensional arrays. In fact, it treats all arrays as one-dimensional, regardless of how they are declared.

Both of the expressions in the "[start, length]" pair are optional. If the "start" value is omitted, the default starting index is 1. If the "length" value is omitted, the default is the length from the starting index to the end of the array. If both are omitted, the default is to use the entire array.

4.7 Array Pointers

The term "array pointer" is a reference to an array. You may think of it as representing the whole array.

Array pointers such as a[] or a$[] can be for assignment and evaluation.

An assignment, such as a[] = b[], makes an exact copy of b[] to a[] including size and dimensions. Background references (i.e passed by referenced variables) are preserved.

Evaluation of a[] always returns 0. Evaluation of a$[] always returns "". Evaluation of non-existent also returns 0 or "".

The behavior of array pointers (e.g. a[]) for assignment and evaluation is different than when used in commands or functions (e.g. myfunc a[]). This is because commands and functions do their own parsing and have their own rules for arrays without indices.

5 Expressions

5.1 Numeric Expression <nexp>

A numeric expression consists of one or more numeric variables or numeric constants separated by binary operators and optionally preceded by unary operators. The definition can be stated more completely using this standard formal notation:

<nexp> := {<numeric variable>|<numeric constant} {<noperator> <nexp>}

The next few sections define all of the terms.

5.1.1 Numeric Operators <noperator>

The numeric operators are listed by precedence. Higher precedence operators are executed before lower precedence operators. Precedence can be changed by using parentheses.

1. Unary +, Unary –
2. Exponent ^
3. Multiply *, Divide /
4. Add +, Subtract –

Note that the comma (',') is not an operator in BASIC!. It is sometimes uses as a separator between expressions; for example, see the **Print** command.

5.1.2 Numeric Expression Examples

- a
- a*b + 4/d – 2*(d^2)
- a + b + d + RND()
- b + CEIL(d/25) + 5

5.1.3 Pre- and Post-Increment Operators

- `++x` Increments the value of x by 1 before the x value is used.
- `--x` Decrements the value of x by 1 before the x value is used.
- `x++` Increments the value of x by 1 after the x value is used.
- `x--` Decrements the value of x by 1 after the x value is used.

```
a = 5      % creates the variable a and sets it to 5
Print –a   % sets a to 4 and prints 4
Print a--  % prints 4 and sets a to 3
```

These operations work only on numeric variables. Their action is performed as part of evaluating the variable, so they do not follow normal precedence rules.

Using these operators on a variable makes the variable unavailable for other operations that require a variable. For example, you cannot pass a variable by reference (see User-Defined Functions) if you pre- or post-increment or -decrement it, because you cannot pass an expression by reference. An exception is made to allow implicit assignment (actual or implied **Let**).

5.2 String Expression <sexp>

A string expression consists of one or more string variables or string constants separated by '+' operators. The definition can be stated more completely using this standard formal notation:

<sexp> := {<string variable>|<string constant>} { + <sexp>}

There is only one string operator: +. This is the concatenation operator. It is used to join two strings:

```
Print "abc" + "def" % prints abcdef
```

5.3 Logical Expression <lexp>

Logical expressions, or Boolean expressions, produce only two results: false or true. False is represented in BASIC! by the numeric value of zero. Anything that is not zero is true. False = 0 and True = not 0.

There are two types of logical expressions: Numeric logical expressions and string logical expressions. Both types produce a numerically-represented values of true or false. Each type consists of one or more variables or constants separated by binary logical operators, formally defined like this:

<slexp> := {<string variable>|<string constant>} <logical operator> {<string variable>|<string constant>}

<nlexp> := {<numeric variable>|<numeric constant>} <logical operator> {<numeric variable>|<numeric constant>}

There is also the unique unary NOT (!) operator. NOT inverts the truth of a logical expression.

5.3.1 Logical Operators

Most of the logical operators are used for comparison. You can compare strings or numbers (<, =, etc.). You can use the other Boolean operators (!, &, |) on numbers but not on strings.

This table shows all of the logical operators. They are listed by precedence with the highest precedence first. All of these operators have lower precedence than any of the numeric operators or the one string operator. Precedence may be modified by using parentheses.

Precedence	Operator	Meaning	Operands
1	< > <= >= = <>	Less Than Greater Than Less Than or Equal Greater Than or Equal Equal Not Equal	Two <nlexp> or two <slexp>
2	!	Unary Not	One <nlexp> only
3	&	And	Two <nlexp> only
4	|	Or	Two <nlexp> only

5.3.2 Examples of Logical Expressions

```
1 < 2 (true)
3 <> 4 (true)
"a" < "bcd" (true)
1 & 0 (false)
!(1 & 0) (true)
```

5.4 Parentheses

Parentheses can be used to override operator precedence.

```
a = b * c + d     % the multiplication is done first
a = b * (c + d)   % the addition is done first
```

Parentheses can also be placed around a variable, anywhere except to the left of an = sign. This can be useful in places where BASIC! may mistake part of a variable for a special keyword. For an example, see the For - To - Step / Next section.

6 Assignment Operations

Variables get values by means of assignment statements. Simple assignment statements are of the form:

```
<nvar> = <nexp>
<svar> = <sexp>
```

The special form of the statement allows BASIC! to infer the command. The implied command is **Let**.

6.1 Let

The original Basic language used the command, **Let**, to denote an assignment operation as in:

```
Let <nvar> = <nexp>
```

BASIC! also has the **Let** command but it is optional. If you use other programming languages, it may look strange to you, but there are two reasons you might use **Let**.

First, you must use **Let** if you want to have a variable name start with a BASIC! keyword. Such keywords may not appear at the beginning of a new line. The statement:

```
Letter$ = "B"
```

is seen by BASIC! as

```
Let ter$ = "B"
```

If you really want to use Letter$ as a variable, you can safely use it by putting it in a **Let** statement:

```
Let Letter$ = "B"
```

If you do the assignment in a single-line **If** statement, you must also use the **Let** command:

```
If 1 < 2 Then Let letter$ = "B"
```

Second, assignment is faster with the **Let** command than without it.

6.2 OpEqual Assignment Operations

All of the binary arithmetic and logical operators (+, -, *, /, ^, &, |) may be used with the equals sign (=) to make a single "OpEqual" operator. The combined operator works like this:

```
var op= expression is the same as var = var op (expression)
```

Here are some examples:

a += 1	is the same as	a = a + 1
a$ += "xyz"	is the same as	a$ = a$ + "xyz"
b /= 5 + 3	is the same as	b = b / (5 + 3)
c ^= log(37) + 1	is the same as	c = c ^ (log(37) + 1)
d *= --d + d--	is the same as	d = d * (--d + d--)
m &= (x$ = y$) \| (x$!= z$)	is the same as	m = m & ((x$ = y$) \| (x$!= z$))

7 Data Structures and Pointers

BASIC! offers commands that facilitate working with Data Structures in ways that are not possible with traditional Basic implementations. These commands provide for the implementation of Lists, Bundles, Stacks and Queues.

7.1 What is a Pointer

The central concept behind the implementation of these commands (and many other BASIC! commands) is the pointer. A pointer is a numeric value that is an index into a list or table of things. Do not confuse the pointer with the thing it points to. A pointer to a List is not a List; a pointer to a bitmap is not a bitmap. A pointer is just a number that represents something else.

As an example of pointers think of a file cabinet drawer with folders in it. That file cabinet is maintained by your administrative assistant. You never see the file drawer itself. In the course of your work you will create a new folder into which you put some information. You then give the folder to your assistant to be placed into the drawer. The assistant puts a unique number on the folder and gives you a slip of paper with that unique number on it. You can later retrieve that folder by asking your assistant to bring you the folder with that particular number on it.

In BASIC! you create an information object (folder). You then give that information object to BASIC! to put into a virtual drawer. BASIC! will give you a unique number—a pointer—for that information object. You then use that pointer to retrieve that particular information object.

Continuing with the folder analogy, let's assume that you have folders that contain information about customers. This information could be things such as name, address and phone number. The number that your assistant will give you when filing the folder will become the customer's customer number. You can retrieve this information about any customer by asking the assistant to bring you the folder with the unique customer number. In BASIC! you would use a Bundle to create that customer information object (folder). The pointer that BASIC! returns when you create the customer Bundle becomes the customer number.

Now let's assume that a customer orders something. You will want to create a Bundle that contains all the order information. Such bundles are used by the order fulfillment department, the billing department and perhaps even the marketing department (to SPAM the customer about similar products). Each Bundle could contain the item ordered, the price, etc. The Bundle will also need to contain information about the customer. Rather than replicate the customer information you will just create a customer number field that contains the customer number (pointer). The pointer that gets returned when you create the order bundle becomes the Order Number. You can create different lists of bundles for use by different departments.

It would also be nice to have a list of all orders made by a customer in the customer Bundle. You would do this by creating a List of all order numbers for that customer. When you create the customer bundle, you would ask BASIC! to create an empty List. BASIC! will return a pointer to this empty List. You would then place this pointer into the customer record. Later when the customer places an order, you will retrieve that list pointer and add the order number to the List.

You may also want to create several other Lists of order Bundles for other purposes. You may, for example, have one List of orders to be filled, another List of filled orders, another List of returned orders, another List for billing, etc. All of these Lists would simply be lists of order numbers. Each order number would point to the order Bundle which would point to the Customer Bundle.

If you were to actually create such a database in BASIC!, you would probably want to save all these Bundles and Lists onto external storage. Getting that information from the internal data structures to external storage is an exercise left to the user for now.

There are things besides List, Bundle, and Stack data structures that are accessed through pointers. These include bitmaps and graphical objects, described in the Graphics chapter, audio clips, described in the SoundPool chapter, and others.

7.2 Lists

A List is similar to a single-dimension array. The difference is in the way a List is built and used. An array must be dimensioned before being used. The number of elements to be placed in the array must be predetermined. A List starts out empty and grows as needed. Elements can be removed, replaced and inserted anywhere within the list.

There is no fixed limit on the size or number of lists. You are limited only by the memory of your device.

Another important difference is that a List is not a variable type. A numeric pointer is returned when a list is created. All further access to the List is by means of that numeric pointer. One implication of this is that it is easy to make a List of Lists. A List of Lists is nothing more than a numeric list containing numeric pointers to other lists.

Lists may be copied into new Arrays. Arrays may be added to Lists.

All of the List commands are demonstrated in the Sample Program file, **f27_list.bas**.

7.3 Bundles

A Bundle is a group of values collected together into a single object. A bundle object may contain any number of string and numeric values. There is no fixed limit on the size or number of bundles. You are limited only by the memory of your device.

The values are set and accessed by keys. A key is a string that identifies the value. For example, a bundle might contain a person's first name and last name. The keys for accessing those name strings could be "first_name" and "last_name". An age numeric value could also be placed in the Bundle using an "age" key.

A new, empty bundle is created by using the **Bundle.create** command. The command returns a pointer to the empty bundle. Because the bundle is represented by a pointer, bundles can be placed in lists and arrays. Bundles can also be contained in other bundles. This means that the combination of lists and bundles can be used to create arbitrarily complex data structures.

After a bundle is created, keys and values can be added to the bundle using the **Bundle.put** command. Those values can be retrieved using the keys in the **Bundle.get** command. There are other bundle commands to facilitate the use of bundles.

7.3.1 Bundle Auto-Create

Every bundle command except **Bundle.create** has a parameter, the <pointer_nexp>, which can point to a bundle. If the expression value points to a bundle, the existing bundle is used. If it does not, and the expression consists only of a single numeric variable, then a new, empty bundle is created, and the variable value is set to point to the new bundle.

That may seem complex, but it isn't, really. If there is a bundle, use it. If there is not, try to create a new one – but BASIC! can't create a new bundle if you don't give it a variable name. BASIC! uses the variable to tell you how to find the new bundle.

```
Bundle.put b,"key1", 1.2
% try to put a value in the bundle pointed to by b
Bundle.put 10, key2$, value2
% try to put a value in the 10th bundle created
Bundle.remove c + d, key$[3],
```

```
% try to remove a key/value pair from a bundle pointed to by c + d
```

In the first example, if the value of **b** points to a bundle, the **Bundle.put** puts **"key1"** and the value **1.2** into that bundle. If **b** is a new variable, its value is 0.0, so it does not point to a bundle. In that case, the **Bundle.put** creates a new bundle, puts **"key1"** and the value **1.2** into the new bundle, and sets **b** to point to the new bundle.

In the second example, if there are at least ten bundles, then the **Bundle.put** tries to put the key named in the variable **key2$** and the value of the variable **value2** into bundle 10. If there is no bundle 10, then the command does nothing. It can't create a new variable because you did not provide a variable to return the bundle pointer.

In the third example, the bundle pointer is the value of the expression **c + d**. If there is no such bundle, the command does nothing. To create a new bundle, the bundle pointer expression must be a single numeric variable.

7.4 Stacks

Stacks are like a magazine for a gun. The last bullet into the magazine is the first bullet out of the magazine. This is also what is true about stacks. The last object placed into the stack is the first object out of the stack. This is called LIFO (Last In First Out).

An example of the use of a stack is the BASIC! **Gosub** command. When a **Gosub** command is executed the line number to return to is "pushed" onto a stack. When a **Return** is executed the return line number is "popped" off of the stack. This methodology allows **Gosubs** to be nested to any level. Any **Return** statement will always return to the line after the last **Gosub** executed.

A running example of Stacks can be found in the Sample Program file, **f29_stack.bas**.

There is no fixed limit on the size or number of stacks. You are limited only by the memory of your device.

7.5 Queues

A Queue is like the line that forms at your bank. When you arrive, you get in the back of the line or queue. When a teller becomes available the person at the head of the line or queue is removed from the queue to be serviced by the teller. The whole line moves forward by one person. Eventually, you get to the head of the line and will be serviced by the next available teller. A queue is something like a stack except the processing order is First In First Out (FIFO) rather than LIFO.

Using our customer order processing analogy, you could create a queue of order bundles for the order processing department. New order bundles would be placed at the end of the queue. The top-of-the-queue bundle would be removed by the order processing department when it was ready to service a new order.

There are no special commands in BASIC! for Queue operations. If you want to make a queue, create a list.

Use **List.add** to add new elements to the end of the queue.

Use **List.get** to get the element at the top of the queue and use **List.remove** to remove that top of queue element. You should, of course, use **List.size** before using **List.get** to ensure that there is a queued element remaining.

III - COMMANDS AND FUNCTIONS

1 App Commands

At the heart of Android devices is an **operating system** that lets many apps operate at the same time. OliBasic is one of the apps. A typical Android device has other apps involving telephone calling, exchange of SMS text messages, operating the device's camera, playing media, and many other activities.

The operating system lets apps send messages to one another. A message is called an **Intent**. The **App.** statements let an OliBasic program send Intents. The **App.broadcast** statement sends an Intent to every other app running on the device. The **App.start** statement sends an Intent to a specific app or to a specific type of apps. The OliBasic program uses these statements to request action from a different app. For example, to play a song, an OliBasic program may send an Intent to a media-player app.

The OliBasic program must be written with knowledge of the other apps on the device, what kind of Intent they will accept, and what information the Intent needs to provide to guide those apps to produce the desired behavior. These details are in the documentation of those apps and are outside the scope of this manual.

1.1 App.broadcast

Syntax: App.broadcast <action_sexp>, data_uri_sexp>, <package_sexp>, <component_sexp>, <mime_type_sexp>, <categories_sexp>, <extras_bptr_nexp>, <flags_nexp>

Creates a system message and broadcasts it to other applications on your device. The message is called an Intent. The Intent will be received by any application that has the right Intent Filter.

All of the parameters are optional; use commas to indicate omitted parameters. See **App.start**, below, for parameter definitions.

If there is no app that can receive your broadcast, the broadcast is ignored. You can detect this condition only by calling the **GetError()** function. If an app is available to receive the broadcast, **GetError$()** returns "No error".

1.2 App.info

Syntax: App.info <package_sexp>, <bundle_pointer_nexp>

Returns information on the application specified by <package_sexp>. If <package_sexp> is **""**, then the package_id of the current application is used.

If you provide a <bundle_pointer_nexp> variable that is not a valid bundle pointer, the command creates a new bundle and returns the bundle pointer in your variable. Otherwise, it writes into the bundle that your variable or expression points to.

The bundle keys and possible values are in the table below:

Key	Type	Value
_NativeLibraryDir	String	Returns the full path to the directory where application's native JNI libraries are stored.
_PackageName	String	package_id.

1.3 App.installed

Syntax: App.installed <flag_nvar>, <package_sexp> {{{,<versionName_svar>}, <versionCode_nvar>}, <pmRaw_svar>}

If the application specified by <package_sexp> is installed, then <flag_nvar> returns 1, else 0. The package should be like "com.rfo.basic"" or "all.sub.My.APP". If present, <versionName_svar> and <versionCode_nvar> are taken from the APK manifest. The package manager returns a raw string list in <pmRaw_svar>.

1.4 App.load

Syntax: App.load <package_sexp>

Loads and installs the application (*.apk) pointed at by the given parse-able URI.

Examples of package expressions:

```
File.root dataPath$

Package$ = "file://" + dataPath$ + "/" + "Bookworm_de.apk"
Package$ = "market://details?id=" + "com.opera.mini.native"
```

Example of an incorrect package expression:

```
Package$ = "http://mougino.free.fr/tmp/920EditorforBASIC_13720.apk"
```

This APK should first be downloaded to the data path. See also the example found in the Sample Program file, f00a_download_manual.bas.

Another example of an incorrect package expression:

```
Package$ = "file://" + "/android_asset/" + "Bookworm_en.apk"
```

This APK is in a protected area. It should first be copied to the data path.

1.5 App.SAR

Syntax: App.SAR <pointer_nexp>

Start **A**nd **R**eceive an App, Intent or Broadcast with a Java like Interface.

For example, FileBrowser, BarCodeScanner, GoogleMaps are supported.

Note: The Blackmoon FileBrowser is suggested, because it does not need the clipboard for multiple files. For more information see http://www.blackmoonit.com/android/filebrowser/.

There is also **App.start** with a different interface and less features, but in some cases it is an option.

The **App.broadcast** command does not work work properly yet, use **Broadcast** (faster) or **App.SAR** for complex uses.

Example:

```
List.create S, commandListPointer
List.add commandListPointer ~
 "new Intent(Intent.ACTION_GET_CONTENT);" ~
 "setPackage(\"com.blackmoonit.android.FileBrowser\");" ~
 "setType(\"*/*\");" ~ % From Ex.: intent.setType( "*/*");
 "addCategory(Intent.CATEGORY_DEFAULT);" ~
 !%From Ex.: intent.addCategory(Intent.CATEG...
```

```
 "addFlags(Intent.FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS);" ~
 ! → );" ~ ← is important
 "EOCL"

List.create S, resultListPointer
List.add resultListPointer "theFilePath$=getData().getPath();", "EORL"

Bundle.create appVarPointer
Bundle.PL appVarPointer, "_CommandList", commandListPointer
Bundle.PL appVarPointer, "_ResultList", resultListPointer
Bundle.put appVarPointer, "theFilePath$", "No file selected!"

App.SAR appVarPointer
```

Note: Java variable, class and function names are case sensitive. In most Java examples → **i**ntent. ← is a variable of the class Intent and should be deleted. Instead → **I**ntent. ← is the class and must not be deleted.

<pointer_nexp> is a bundle pointer. The bundle can have the following keys:

_CommandList is a required key expression.

_ResultList is a required key expression if you want results.

_Broadcast is a required key expression if you want to send a Broadcast.

Variables used in the _CommandList (only the bracketed variables with exclamation like “!variable$!”) and _ResultList have to be put in the transfer bundle, too.

Note: The maximum size for the intent bundle is limited to about 1 MB when transferring data with intents.

TIP: If the expected results are not returned, then PRINT the expressions to debug your code.

Supported:

- new Intent
- setAction
- setPackage
- putExtra only Bundle, String, Long, Double and the arguments true and false
- addCategory
- addFlags
- setComponent(new ComponentName)
- setData
- setDataAndType
- setType

Caution: If you want to set both the URI and MIME type, don't call setData() and setType() because they each nullify the value of the other. Always use setDataAndType() to set both URI and MIME type.

- getStringExtra
- getIntExtra
- getDoubleExtra
- getBundleExtra
- getBooleanExtra
- getData().getPath()
- getLaunchIntentForPackage()
- getParcelableArrayListExtra(Intent.EXTRA_STREAM)

- Out of The box
- createChooser
- "createChooser("+ CHR$(34) + "Chooser Title, share with ..." + CHR$(34) +");" ~

This list may be expanded. Stay tuned!

In the underlying Java machine, two main types of application starts were implemented.

First type: **startActivity** used by **App.start** and **App.SAR** without _ResultList

In this case the activity is **not launched** as a **sub**-activity!

Second type: **startActivityForResult** used by **App.SAR** with _ResultList

Important: If you apply **App.SAR** with _ResultList then use the suggested ReorderToFront() function below before finishing, which prevents trouble if you hit the home button at sub intent runtime. In this case **Console.front** and **Gr.front** would fail.

If the called intent was written in Java or a language other than BASIC!, use the appropriate commands.

A look at https://developer.android.com/guide/components/intents-filters is recommended.

See also **Bundle.In**, **Bundle.Out**.

Examples:

```
!------------- ReorderToFront() also part of TestTest.bas -------------
Fn.def ReorderToFront()
 Program.info b
 Bundle.get b, "_PackageName", pN$
 List.create S, commandListPointer
 List.add commandListPointer ~
  "new Intent(Intent.ACTION_MAIN);" ~
  "setPackage("+ CHR$(34) + pN$ + CHR$(34) +");" ~
  "addCategory(Intent.CATEGORY_DEFAULT);" ~
  "addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);" ~
  "EOCL"
 Bundle.PL appVarPointer,"_CommandList",commandListPointer
 App.SAR appVarPointer
 Fn.rtn appVarPointer
Fn.end

!----------------- Rest of TestTest.bas -------------------
Program.info bPointer
Bundle.get bPointer, "_BasName", bN$
Bundle.get bPointer, "_PackageName", pN$
Bundle.get bPointer, "_MyProcessId", mpid$
Console.title bN$ + "   runs on the  " + pN$ + "   Basic Engine"
? "_MyProcessId", mpid$
File.root fp$,"_Programpath"
? fp$
Bundle.in action$, data$, bi
? "action$ ";action$
? "data$ "; data$
p = 2000
? "Waiting " + Int$(p/1000) + " seconds before killing"
Pause p
Bundle.put resultPointer, "EXTRA_RESULT", "EXTRA_RESULT Strings Test"
Bundle.put resultPointer, "Test", "Strings Test"
Bundle.put resultPointer, "TestNum", 4711
Bundle.out a$, d$, resultPointer
!To prevent a Home-button-hitting-issue if you need automatic itself-returning
```

```
ReorderToFront()
Exit
```

In the next examples, if you use the variable basicEngine$ you can use:

com.rfo.basicOli ,
com.rfo.basicFellow, NEW
com.rfo.basic, with limitations

Or crate your own with:

```
Bundle.in action$, data$, bi
File.exists ok, data$
If ok Then Run data$
sel = -4000  % If negative, the dialog.message will close
             % after <sel> Milliseconds
Dialog.message  "Program File not Found!", data$, sel
Exit
```

If you call a second BASIC! engine start (maybe with com.rfo.basicFellow) from OliBasicFellow**.apk and call a **different** engine (maybe com.rfo.basicOli):

```
!----------------- SubIntentWithResult.bas -------------------
Fn.def SubIntentWithResult(basEngine$, basProgramPath$, mode, testMode)
 eMode$ = ""
 If mode > 0 Then eMode$ = "_Editor"
 List.create S, commandListPointer
 List.add commandListPointer ~
  "new Intent(Intent.ACTION_MAIN);" ~
  "setData("+ Chr$(34) + basProgramPath$ + Chr$(34) + ");" ~
  "new ComponentName(\"" + basEngine$ + "\"" + "," + "\"" + ~
  basEngine$ + ".Basic" + "\"" + ");" ~
  "addCategory(Intent.CATEGORY_EMBED);" ~
  !Starts program in Editor mode, if eMode$ = "_Editor"!
  "putExtra(" + Chr$(34) + "_BASIC!" + Chr$(34) + ","+ Chr$(34) + ~
  eMode$ + Chr$(34) + ");" ~
 "addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);" ~
 "EOCL"

 List.create S, resultListPointer
 If testMode > 0 Then
  List.add resultListPointer ~
   "theDataPath$=getData().getPath();" ~
   "EXTRA_RESULT$=getStringExtra(" + Chr$(34) + "EXTRA_RESULT" + ~
   Chr$(34) + ");" ~
   "Test$=getStringExtra(" + Chr$(34) + "Test" + Chr$(34) + ");" ~
   "TestNumN=getDoubleExtra(" + Chr$(34) + "TestNum" + Chr$(34) + ");" ~
   "EORL"
 Else
  List.add resultListPointer "EORL"
 EndIf

 Bundle.PL appVarPointer, "_CommandList", commandListPointer
 Bundle.PL appVarPointer, "_ResultList", resultListPointer
 If testMode > 0 Then
  Bundle.Put appVarPointer, "theDataPath$", basProgramPath$
  Bundle.Put appVarPointer, "EXTRA_RESULT$", EXTRA_RESULT$
  Bundle.Put appVarPointer, "Test$", Test$
  Bundle.Put appVarPointer, "TestNumN", TestNum
 ENDIF

 APP.SAR appVarPointer
 Fn.rtn appVarPointer
Fn.end

Fn.def ReturnedItems$(appVarPointer, pr)
```

```
 Bundle.keys appVarPointer, keyListPointer
 List.size keyListPointer, listSize
 If pr
  ? "listSize ", listSize - 4
  For I = 1 TO listSize
   List.get keyListPointer, I, item$
   If item$ = "_CommandList" | item$ = "_ResultList" | ~
    item$ = "%succededResults%" | item$ = "DebugSI"
   Else
    Print item$
   EndIf
  Next i
 EndIf
 Bundle.get appVarPointer, "%succededResults%", Results$
 Fn.rtn Results$
Fn.end

basEngine$ = "com.rfo.basicOli" % Your favorite BASIC! engine
File.root fp$ ,"_External" % Start point of the public part of
                          % the external file system
basProgramPath$ = "file://" + fp$ + "/rfo-basic/source/TestTest.bas"
mode = 0                  % Only execution mode
! mode = 1 %Editor mode
! testMode = 0 % Without results
testMode = 1 % With results
appVarPointer = SubIntentWithResult(basEngine$, basProgramPath$, mode, testMode)
If testMode
 myReturnedItems$ = ReturnedItems$(appVarPointer, 0)
 Print myReturnedItems$
 Bundle.get appVarPointer,"EXTRA_RESULT$", EXTRA_RESULT$
 Print "EXTRA_RESULT=: "; "<" + EXTRA_RESULT$ + ">"
 Bundle.get appVarPointer, "Test$", Test$
 Print "Test=: "; "<" + Test$ + ">"
 Bundle.get appVarPointer, "TestNumN", TestNum
 Print "TestNum=: "; "<"; TestNum; ">"
EndIf
End


!----------------- IndependentLaunch.bas -------------------
Fn.def IndependentLaunch(basEngine$, basProgramPath$, mode)
 eMode$ = ""
 If mode > 0 Then eMode$ = "_Editor"
 List.create S, commandListPointer
 List.add commandListPointer ~
  "new Intent(Intent.ACTION_MAIN);" ~
  "setData(" + Chr$(34) + basProgramPath$ + Chr$(34) + ");" ~
  "new ComponentName(\"" + basEngine$ + "\"" + "," + "\"" + ~
  basEngine$ + ".Basic" + "\"" + ");" ~
  "addCategory(Intent.CATEGORY_LAUNCHER);" ~
  "putExtra(" + Chr$(34) + "_BASIC!" + Chr$(34) + ","+ Chr$(34) + ~
  eMode$ + Chr$(34) + ");" ~
  "addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);" ~
  ! vv
  "addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);" ~
  "addFlags(Intent.FLAG_ACTIVITY_MULTIPLE_TASK);" ~
  "addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);" ~
  ! ^^
  "EOCL"
 Bundle.PL appVarPointer,"_CommandList", commandListPointer
 App.SAR appVarPointer
Fn.end

basEngine$ = "com.rfo.basicOli" % You favorite BASIC! engine
File.root fp$ , "_External"     % Start point of the public part of the
                                % external file system
basProgramPath$ = "file://" + fp$ + "/rfo-basic/source/TestTest.bas"
! mode = 1 % Program starts in BASIC!'s Editor
mode = 0
```

```
Call IndependentLaunch(basEngine$, basProgramPath$, mode)
End
```

1.6 App.settings

Syntax: App.settings {<package_sexp>}

Calls the application settings of the app with package-id specified by the optional <package_sexp>. If the package-id is not given, application settings of the running BASIC! engine or the settings of the executed application created with BASIC! is opened.

App.settings is asynchronous on runtime. Use **Dialog.message** or **Dialog.select** to suspend program execution.

Example:

```
App.settings % "com.rfo.basicOli"
Dialog.message "Settings", "Shoud we check again?", sel, "Yes", "No"
If sel = 1 Then GetPernissions()
```

1.7 App.start

Syntax: App.start <action_sexp>, data_uri_sexp>, <package_sexp>, <component_sexp>, <mime_type_sexp>, <categories_sexp>, <extras_bptr_nexp>, <flags_nexp>

Sends to the system a message, called an Intent, requesting a specific application or type of application to start. If more than one app can handle the request, the system puts up a chooser for you.

If there is no app that can handle the request, the Intent is ignored. You can detect this condition only by calling the **GetError$()** function. If an app is available to launch, **GetError$()** returns "No error".

All of the parameters are optional. Use commas to indicate omitted parameters. You will almost never need to use all of the parameters in one command.

The first six parameters are string expressions: action, data URI, package name, component name, MIME type, and a list of categories separated by commas (the commas are part of the string expression).

The last two parameters are numeric expressions. One is a pointer to a bundle that contains "extras" that are attached to the message. The other is a single number representing one or more flag values.

For parameter values, consult the documentation of the Android system and the app you want to start.

Parameter	Meaning	am Command Equivalent
action	An Action defined by Android or by the target application	-a
data URI	Data or path to data; BASIC! URI-encodes this string	-d
package name	Name of the target application's package (sometimes called ID)	-n [Note 2]
component name	Name of a component within the target application	-n [Note 2]
MIME type	MIME-type of the data, may be used without a data URI	-t
categories	Comma-separated list of Intent categories	-c [Note 3]
"extras" bundle ptr	Pointer to an existing bundle containing **"**extras**"** values	-e [Note 4]

flags	Sum of numeric values of one or more flags	-f

Notes:

1. You must use string or numeric values, not Android-defined constants, for actions, types, categories, and flags. For example, you can use the string **"android.intent.action.MAIN"**, but you can not use the Android symbol **ACTION_MAIN**.

2. If you specify a component name, you must also specify the package name, even though the package name is often part of the component name. For example, these are equivalent:

```
System.write "am -n com.android.calculator2.Calculator"
App.start , , "com.android.calculator2", "com.android.calculator2.Calculator"
```

   Usually you can use this pattern:

```
pkg$ = "com.android.calculator2" : comp$ = pkg$ + ".Calculator"
App.start , , pkg$, comp$
```

3. A categories parameter may contain several categories separated by commas, for example:

```
cats$ = "android.intent.category.BROWSABLE, android.intent.category.MONKEY"
cats$ = cat1$ + "," + cat2$ + "," + cat3$
```

4. You must create and populate the "extras" bundle before passing its pointer to an **App** command. Presently, only string extras and float extras are supported (the BASIC! data types).

5. When your program uses an Intent to start another app, the second app can use another Intent to return results. BASIC! does not yet support this return path. Your program can retrieve data that another app leaves in a file or in the clipboard, but it cannot yet retrieve data from a return Intent.

1.8 Browse

Syntax: Browse <url_sexp>

If <url_sexp> starts with "http…" then the internet site specified by <url_sexp> will be opened and displayed.

If <url_sexp> starts with "file://" + absolutePath$ + "/" + fileName$ then the file will be open by a linked application. Starting with Android 11+ the File Provider is needed for nonpublic files. See PROVIDER.

Note: You can also use the HTML commands to display (and interact with) web pages located on your device or on the web.

See also: **Gr.camera.takeVideo**.

2 Array Commands

See description of arrays in section on Arrays.

2.1 Array.average

Syntax: Array.average <Average_nvar>, Array[{<start>, <length>}]

Finds the average of the values in a numeric array (Array[]) or array segment (Array[start, length]), and places the result into <Average_nvar>.

2.2 Array.binary.search

Syntax: Array.binary.search Array[]|Array$[], <value_exp>, <result_nvar>

Searches the numeric or string array (Array[] or Array$[]) for the specified numeric or string value, which may be an expression. If the value is found in the array, its position will be returned in the result numeric variable <result_nvar>. If the value is not found the result will be zero.

This command uses the very fast binary-search-method. The array should have already been sorted or there may be unexpected results.

Example:

```
Array.sort myTownArray$[]
Array.binary.search myTownArray$[], "Paris", resultIndex
```

2.3 Array.by.index

Syntax: Array.by.index SourceArray[], IndexArray[], DestinationArray[]

or

Syntax: Array.by.index SourceArray$[], IndexArray[], DestinationArray$[]

Copies elements of existing SourceArray[] to DestinationArray[], as specified by IndexArray[].

Example:

```
Array.load values[], 1, 2.45, 3, 4, 5, 6
Array.load idx[], 1, 4, 2, 3, 1, 5, 2, 6, 2
Array.by.index values[], idx[], out[]
Join.all out[], result$, ", "
Print result$      % Returns "1, 4, 2.45, 3, 1, 5, 2.45, 6, 2.45"
```

2.4 Array.copy

Syntax: Array.copy SourceArray[{<start>,<length>}],
DestinationArray[{{-}<start_or_extras_nexp>}]

Copies elements of an existing SourceArray[] to the DestinationArray[]. If the Destination Array exists, some or all of the existing array is overwritten. If the Destination Array does not exist, a new array is created. The arrays may be either numeric or string arrays but they must both be of the same type.

You may copy an entire array (SourceArray[]) or an array segment (SourceArray[<start>,<length>]). Copying stops without error if it reaches the end of either the SourceArray or the DestinationArray.

If <start> is < 1 it is set to 1, the first element of the SourceArray. If <length> is < 0 it is set to 0.

If the Destination Array already exists, the optional <start_or_extras> parameter specifies where to start copying into the Destination Array.

If the Destination Array does not exist, the optional <start_or_extras_nexp> parameter specifies how many extra elements to add to the copy. If the parameter is a negative number, these elements are added to the start of the array, otherwise they are added to end of the array.

The extra elements for a new numeric array are initialized to zero. The extra elements for a new string array are initialized to the empty string, "".

See the Sample Program file, f26_array_copy.bas, for working examples of this command.

Note: The destination array will always be a flattened vector with a single dimension. The statement **B[] = A[]** will also perform an array copy, but the dimensions of the destination array will be the same as those of the source array.

2.5 Array.delete

Syntax: Array.delete Array[]{, Array[]} ...

Does the same thing as **UnDim** Array[].

2.6 Array.dims

Syntax: Array.dims Source[]{, {Dims[]}{, NumDims_nvar}}

Provides information about the dimensions of the Source[] array parameter. The Source[] parameter may be a numeric or string array name with nothing in the brackets ("[]"). The array must already exist. The Source[] parameter is required, and both of the other parameters are optional.

The dimensions of the Source[] array are written to the Dims[] array, if you provide one. The Dims[] parameter must be a numeric array name with nothing in the brackets ("[]"). If the Dims[] array exists, it is overwritten. Otherwise a new array is created. The result is always a one-dimensional array.

The number of dimensions of the Source[] array is written to the NumDims_nvar parameter, if you provide one. NumDims_nvar must be a numeric variable. This value is the length of the Dims[] array.

2.7 Array.fill

Syntax: Array.fill Array[{<start>,<length>}], <exp>

Fills an existing array or array segment with a value. The types of the array and value must match.

2.8 Array.fromstring

Syntax: Array.from.string <sexp>, Array[]

Converts characters of strings into an array of numbers like the **Ucode()** function.

Example:

```
Array.from.string s$, nA[]
```

2.9 Array.length

Syntax: Array.length <length_nvar>, Array[{<start>,<length>}]

Places the number of elements in an entire array (Array[] or Array$[]) or an array segment (Array[start, length] or Array$[start, length]) into <length_nvar>.

2.10 Array.load

Syntax: Array.load Array[], <exp>, ...

Creates a new array, evaluates the list of expressions "<exp>, ...", and loads values into the new array. Specify the array name with no index(es). The array has one dimension; its size is the same as the number of expressions in the list. If the named array already exists, it is overwritten.

The array may be numeric (Array[]) or string (Array$[]), and the expressions must be the same type as the array.0

The list of expressions may be continued onto the next line by ending the line with the "~" character. The "~" character may be used between <exp> parameters, where a comma would normally appear. The "~" itself separates the parameters; the comma is optional.

The "~" character may not be used to split a parameter across multiple lines.

Examples:

```
Array.load Numbers[], 2, 4, 8 , n^2, 32
Array.load Hours[], 3, 4,7,0, 99, 3, 66~  % comma not required before ~
                    37, 66, 43, 83,~      % comma is allowed before ~
                    83, n*5, q/2 +j
Array.load Letters$[], "a", "b","c",d$,"e"
```

2.11 Array.max

Syntax: Array.max <max_nvar>, Array[{<start>, <length>}]

Finds the maximum value in a numeric array (Array[]) or array segment (Array[start, length]), and places the result into the numeric variable <max_nvar>.

2.12 Array.median

Syntax: Array.median <median_nvar>, Array[]

Returns the median of an array in <median_nvar>. The median is the value in the middle of the given (temporarily) sorted array. If the length of the array is even, the average of the two middle values is returned.

2.13 Array.min

Syntax: Array.min <min_nvar>, Array[{<start>, <length>}]

Finds the minimum value in a numeric array (Array[]) or array segment (Array[start, length]), and places the result into the numeric variable <min_nvar>.

2.14 Array.reverse

Syntax: Array.reverse Array[{<start>, <length>}]

Reverses the order of values in a numeric or string array (Array[] or Array$[]) or array segment (Array[start, length] or Array$[start, length]).

2.15 Array.rnd

Syntax: Array.rnd Array[] {{{{{{, <length_nexp>}, <low_nexp>}, <high_nexp>}, <seed_nexp>}, <type_nexp>}, <generator_nexp>}

Creates a numeric array with pseudo-random numbers by using a new random generator. The array length is described by the optional <length_nexp>. If not given, an array with only one member is returned. The optional <low_nexp> sets the starting value or the offset for Gaussian random numbers. Default is 0. The optional <high_nexp> sets the ending value or the scale for Gaussian random numbers. Default is 1. If <low_nexp> = 0, <high_nexp> is the scale factor in conjunction to RND(). The optional <seed_nexp> sets the seed. Default is 0 for unpredictable and not reproducible numbers. Seed number > 0 creates reproducible results.

The optional <type_nexp> sets the returned type of the random numbers. Default is 0 for normal random numbers. If <type_nexp> is 1, Gaussian random numbers are returned. I f <type_nexp> is 2, boolean random numbers are returned.

The optional <generator_nexp> sets the random generator. Default is 0 for the standard generator. If <generator_nexp> is 1, a secure random number generator (RNG) implementing the default random number algorithm will be constructed. Only usable with Android 4.4+ (KitKat+).

Example 1:

```
Array.Rnd r[], 2, 0, 100, 4711, 0
Print r[1]  % Returns 4.315872869138159
Print r[2]  % Returns 37.83154009540295
Array.Rnd nr[]
Print nr[1]  % Returns unpredictable, not reproducible number between 0 and < 1.
```

Example 2:

```
offs=20 : fac=10
Array.Rnd  rnd[],1, offs, offs + fac
! The results will be between 20 and 30
Print rnd[1], offs + RND() * fac
Array.Rnd  rnd[],1, offs, offs + fac, 0, 2 % Boolean result
! Now the result will be 20 or 30
Print rnd[1]
```

2.16 Array.row.print

Syntax: Array.row.print SourceArray[], {{<lineNum_nexp>}, <result_svar>

Prints an array as rows into the console or a string. The order is defined by the given dimensions of the array. If <lineNum_nexp> is grater than 0, line numbers are added at the start of the row. Default is 1. If <result_svar> is given, the rows are transferred into a string with line feed endings ("\n"). In this case no console output is returned.

Example:

```
Array.load xyzPoints[], 0,0,0, 250,100,0, 500,0,0, 500,500,0, 250,400,0,
0,500,0
! Returns rows with line numbers and x, y, z values → 1: 0.0, 0.0, 0.0 ...
Array.row.print  xyzPoints[], 1
```

2.17 Array.search

Syntax: Array.search Array[{<start>,<length>}], <value_exp>, <result_nvar>{,<start_nexp>}

Searches in the numeric or string array (Array[] or Array$[]) or array segment (Array[start, length] or Array$[start, length]) for the specified numeric or string value, which may be an expression. If the value is found in the array, its position will be returned in the result numeric variable <result_nvar>. If the value is not found the result will be zero.

If the optional start expression parameter is present, the search will start at the specified element. The

default value is 1.

If only a segment of an array is used, the result will be relative to the start point.

Example:

```
Array.load ar[],10, 8,12, 20, 6, 7, 88, 11
Array.search ar[4, 3], 6, pos          % pos = 2 instead of 5
Array.search ar[4, 3], 88, pos         % pos=0, because 88 is
                                       % not part of the segment
```

2.18 Array.shuffle

Syntax: Array.shuffle Array[{<start>, <length>}]

Randomly shuffles the values of the specified array (Array[] or Array$[]) or array segment (Array[start, length] or Array$[start, length]).

2.19 Array.sort

Syntax: Array.sort Array[{<start>, <length>}]

Sorts the values of the specified array (Array[] or Array$[]) or array segment (Array[start, length] or Array$[start, length]) in ascending order.

2.20 Array.std_dev

Syntax: Array.std_dev <sd_nvar>, Array[{<start>, <length>}]

Finds the standard deviation of the values in a numeric array (Array[]) or array segment (Array[start, length]), and places the result into the numeric variable <sd_nvar>.

2.21 Array.sum

Syntax: Array.sum <sum_nvar>, Array[{<start>, <length>}]

Finds the sum of the values in a numeric array (Array[]) or array segment (Array[start, length]), and then places the result into the numeric variable <sum_nvar>.

2.22 Array.to.dims

Syntax: Array.to.dims SourceArray[], DimensionArray[], DestinationArray[]

Copies all elements of existing SourceArray[] to DestinationArray[], as specified in order by DimensionArray[]. The number of source and destination array elements must be the equal. If the DestinationArray exists, it will be overwritten. If the DestinationArray does not exist, a new array is created. The source and destination arrays may be either numeric or string arrays but they must both be of the same type. The dimension array must be numeric type.

Example:

```
Array.load s[], 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 % 10 elements
Array.load d[], 5, 2 % 5 * 2 = 10
Array.to.dims s[], d[], r[]
Print r[1, 1]  % Returns 0.0
Print r[5, 2]  % Returns 9.0
```

\	1	2	3	4	5
1	*0*	*2*	*4*	*6*	*8*

2	*1*	*3*	*5*	*7*	*9*

2.23 Array.to.string

Syntax: Array.to.string <sexp>, Array[]

Converts the numbers of a numeric array into a string, like the CHR$() function.

Example:

```
Array.to.string s$, nA[]
```

2.24 Array.truth.choice

Syntax: Array.truth.choice SourceTrueArray[], SourceFalseArray[], IndexArray[], DestinationArray[]

Copies elements of existing SourceTrueArray[] or SourceFalseArray[] to the DestinationArray[], depending on the specifications of the IndexArray[]. If the corresponding item of the IndexArray[] is 0, the item of the SourceFalseArray$[] will be copied. Otherwise, if the item in the IndexArray[] is not 0, the SourceTrueArray$[] item will be copied.

IndexArray[], SourceTrueArray[] and SourceFalseArray[] must have the same number of items. SourceTrueArray[], SourceFalseArray[] and DestinationArray[] must all be the same type, numeric or string.

Example:

```
Array.load valuesTrue[], 1, 2.45, 3, 4, 5, 6
Array.load valuesFalse[], 6, 5, 4, 3, 2.45, 1
Array.load idx[], 1, 0, 0, 0, 0, 1
Array.truth.choice valuesTrue[], valuesFalse[], idx[], out[]
Join.all out[], result$, ", "
Print result$   % Returns "1, 5, 4, 3, 2.45, 6"
```

2.25 Array.truth.index

Syntax: Array.truth.index <is_truth_nexp>, IndexArray[], DestinationArray[]

Copies elements of existing IndexArray[] to DestinationArray[], as specified by IndexArray[]. If <is_truth_nexp> is 0, all index entries in conjunction to IndexArray[] members which are 0 are copied. If <is_truth_nexp> is not 0, all source elements in conjunction to IndexArray[] members which are not 0 are copied.

Example:

```
Array.load in2[], 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0
Array.truth.index 1, in2[], out[]
! -> out[] will be: 3, 6, 9, 11, 12
Array.truth.index 0, in2[], out[]
! -> out[] will be: 1, 2, 4, 5, 7, 8, 10, 13
```

2.26 Array.truth.subset

Syntax: Array.truth.subset <is_truth_nexp>, SourceArray[], IndexArray[], DestinationArray[]

Copies elements of an existing SourceArray[] to the DestinationArray[], which are specified by the IndexArray[]. If <is_truth_nexp> is 0, all source entries in conjunction to IndexArray[] members which are 0 are copied. If <is_truth_nexp> is not 0, all source elements in conjunction to IndexArray[] members which are not 0 are copied.

SourceArray[] and DestinationArray[] must be the same type, numeric or string.

Example:

```
Array.load in1[], 1,2,27,4,5,6,7,8,3,4,57,114,115
Array.load in2[], 0,0,1 ,0,0,1,0,0,1,0,1 ,1  ,0
Array.truth.subset 1, in1[],in2[],out[]
! -> out[] will be: 27, 6, 3, 57, 114
Array.truth.subset 0, in1[],in2[],out[]
! -> out[] will be: 1, 2, 4, 5, 7, 8, 4, 115
Array.load in3$[], "1", "2", "27", "4", "5", "6", "7", "8", "3", "4", "57",
"114", "115"
Array.truth.subset 0, in3$[],in2[],out2$[]
! -> out2$[] will be: 1, 2, 4, 5, 7, 8, 4, 115
```

2.27 Array.variance

Syntax: Array.variance <v_nvar>, Array[{<start>,<length>}]

Finds the variance of the values in a numeric array (Array[]) or array segment (Array[start,length]), and places the result into the numeric variable <v_nvar>.

2.28 Dim

Syntax: Dim Array[<nexp>{, <nexp> } ...] {, Array[<nexp>{, <nexp> } ...] } ...

The **Dim** command specifies how many dimensions an array will have and how big those dimensions are. The array is created reserving and initializing memory for the array data. All elements of a numeric array are initialized to the value 0.0. String array elements are initialized to the empty string, "". If you **Dim** an array that already exists, the existing array is destroyed and a new one created.

Multiple arrays can be dimensioned with one **Dim** statement. String and numeric arrays can be dimensioned in a single **Dim** command.

Examples:

```
Dim A[15]
Dim B$[2,6,8], C[3,1,7,3], D[8]
```

2.29 ReDim

Syntax: ReDim {<preserve_nexp>}, Array[<nexp>{, <nexp> } ...] ...

Re-Dimensions an existing array with optional 'preserve contents' flag. Referenced arrays are not cut off (when passing to a user function). Arrays must already exist.

If the preserve flag is zero, the array will be full of 0's or "". If the preserve flag is non-zero, the old contents will be preserved to the size of the old or new array, whichever was smallest. Any cells left over will be 0's or "".

Preserve flags may be inserted anywhere in the arguments list. Arrays following a preserve flag will honor that preserve flag until the next preserve flag. Preserve flags cannot be array expressions e.g. p[3], as this would re-dim p[3].

Example:

```
ReDim 1, a[], 0, b[] 1, c[], d[]     % b[] will not be preserved
```

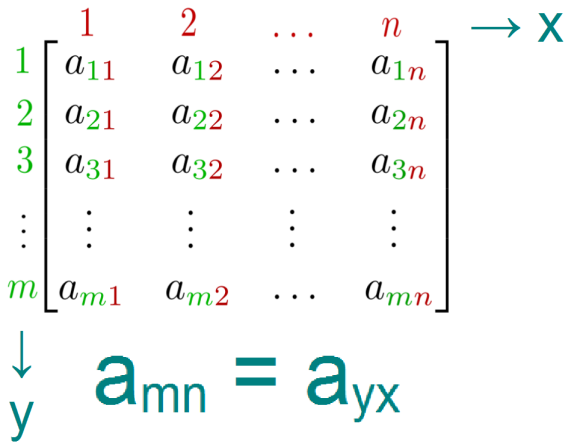
2.30 UnDim

Syntax: UnDim Array[]{, Array[] } ...

“Undimensions” an array. The array is destroyed, releasing all of the memory it used. Multiple arrays can be destroyed with one **UnDim** statement. Each Array[] is specified without any index. This command is exactly the same as **Array.delete**.

3 Array Matrix Commands

Array matrix commands start with **Array.mat**, where mat stands for matrix. A matrix is a two-dimensional array. Usually, matrices are shown with capital letters such as **A** or **B**, number of columns with **n** and number of rows with **m**.



An m × n matrix: the m rows are horizontal and the n columns are vertical. Each element of a matrix is often denoted by a variable with two subscripts. For example, a2,1 represents the element at the second row and first column of the matrix.

Source: Wikipedia

Example:

```
Array.load s[], 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 % 10 elements
Array.load d[], 5, 2 % 5 * 2 = 10
Array.to.dims s[], d[], r[]
```

r [] → [5,2] is a matrix defined by columns (5) and rows (2), **but the dimension positions of *rows* "y" and *columns* "x" are reversed contrary to the mathematical notation *rows* "n" and *columns* "m".** The order in BASIC! maybe different to other programming languages, but is the same as in FORTRAN. The advantage is that the result corresponds to the order ([width (x), height (y)]) of the bitmap or screen pixels.

→ x
↓
y

\	**1**	**2**	**3**	**4**	**5**
1	*0*	*2*	*4*	*6*	*8*
2	*1*	*3*	*5*	*7*	*9*

If you need an order like the example below, use the _Toggle skill of **Array.Mat.Skill**, or **Array.Mat.Toggle**.

\	**1**	**2**	**3**	**4**	**5**
1	*0*	*1*	*2*	*3*	*4*
2	*5*	*6*	*7*	*8*	*9*

Array.copy r [1,4], c[]

Equals to **Array.copy s** [1,4], c[] because it is flattened to a vector and looses its dimensions except for one.

c [] → [4] it is a Vector, it has only one dimension, it has by definition no columns or rows.

\	1	2	3	4	
1	*0*	*1*	*2*	*3*	*?*
2	*1*				
3	*2*				
4	*3*				
	?				

If the input needs a Matrix, the Vector elements are placed as a **column** by default.

c [] → C

\	1
1	*0*
2	*1*
3	*2*
4	*3*

If you need a Matrix **row** input use the _Transpose skill of **Array.Mat.Skill**, or **Array.Mat.Transpose**.

\	1	2	3	4
1	*0*	*1*	*2*	*3*

A Scalar is a special case of a Vector with only one element. In a Matrix it has only one column and one row.
C[4] →

\	1
1	*3*

Single numeric values are automatically converted into a Scalar, thus it is processed as an array with only one element.

3.1 Array.mat.toggle

Syntax: Array.mat.toggle SourceArray[], {<direction_sexp>}, DestinationArray[]

Copies all elements of existing SourceArray[] to DestinationArray[], but toggles the column and row order over the element [1,1].

The direction can be specified by the optional <direction_sexp>.

- With the character string "_CtoR" a **column** packed array can be copied into a **row** packed array. This is the default direction.
- With the character string "_RtoC" a **row** packed array can be copied into a **column** packed array.

The sum of the source and destination array elements is the same. If the destination array is present, it will be overwritten. If the destination array does not exist, a new array will be created. The arrays can be either numeric or string arrays, but both must be of the same type.

In the case of square matrices, the results of **Array.mat.toggle** and **Array.mat.transpose** are the same.

Example:

```
ARRAY.load s[], 0, 1, 2, 3, 4, 5, 6, 7, 8, 9  % 10 elements
Array.load d[], 5, 2 % 5 * 2 = 10
Array.to.dims s[], d[], r[]
Array.mat.toggle r[], "_CtoR", n[]
Array.dims n[], d[] % d[] returns 5, 2 as usual
```

Before toggling:

\	**1**	**2**	**3**	**4**	**5**
1	*0*	*2*	*4*	*6*	*8*
2	*1*	*3*	*5*	*7*	*9*

After toggling:

\	**1**	**2**	**3**	**4**	**5**
1	*0*	*1*	*2*	*3*	*4*
2	*5*	*6*	*7*	*8*	*9*

3.2 Array.mat.transpose

Syntax: Array.mat.transpose SourceArray[], DestinationArray[]

Copies all elements of existing SourceArray[] to DestinationArray[], but transposes over the element [1,1]. The sum of the source and destination array elements is the same. If the destination array is present, it will be overwritten. If the destination array does not exist, a new array is created. The arrays can be either numeric or string arrays, but both must be of the same type.

In the case of square matrices, the results of **Array.mat.toggle** and **Array.mat.transpose** are the same.

Example:

```
Array.load s[], 0, 1, 2, 3, 4, 5, 6, 7, 8, 9  % 10 elements
Array.load d[], 5, 2 % 5 (Columns) * 2 (Rows) = 10
Array.to.dims s[], d[], r[]

! How to store the third column of Array r[] (Matrix) into a list

columnInArrayR = 3 : nRowsInArrayR = 2 % In r[] !
List.create n, vertical
List.add.array vertical, r[1 + (columnInArrayR -1) * nRowsInArrayR, ~
 nRowsInArrayR]

! How to store the second row of Array r[] (Matrix) into a list

Array.mat.transpose r[], n[]
Array.dims n[], d[] % d[] returns 2, 5 instead 5, 2
```

```
rowInArrayR = 2 : nColumnsInArrayR = 5 % In r[] !
List.create n, horizontal
List.add.array horizontal, n[1 + (rowInArrayR -1) * nColumnsInArrayR, ~
 nColumnsInArrayR]
```

Before transposing:

\	1	2	3	4	5
1	0	2	4	6	8
2	1	3	5	7	9

After transposing:

\	1	2
1	0	1
2	2	3
3	4	5
4	6	7
5	8	9

3.3 Array.mat.skill

Syntax: Array.mat.skill <bundle_pointer_nexp>{, <runtime_error_nexp>}

Arguments and results are stored in the bundle specified by <bundle_pointer_nexp>.

The bundle key "_Result" stores the result.

The bundle key "_Error" logs errors if any occur.

If <runtime_error_nexp> specified a value > 0, the execution is aborted immediately in a case of an error. The default is 1. If the value is 0, the execution will not stopped if possible.

LEVEL 1

A skill is specified by a bundle entry like this

```
Bundle.put sBptr, "_Skill", "_min('ArrayLeft', 'ArrayRight')"
```

The operator begins with an underscore, and its arguments are within parentheses.

The key names of the arguments must be within single quotes.

The following four characters `):('` must not be between single quotes.

```
Bundle.put sBptr, "ArrayLeft",  mArrayLeft[]
Bundle.put sBptr, "ArrayRight",  mArrayRight[]
```

After successful execution a result is returned by the bundle key "_Result".

```
Bundle.get sBptr, "_Result",  mResult[]
```

Otherwise this key is deleted or not created. In this case the bundle key "_Error" returns the error log.

The result of an operation can be a numeric vector (one-dimensional array) or a matrix (two-dimensional array). If a single value is the default result, a one-dimensional array with length 1 is returned.

All possibilities of **Array.math** are supported, thus operations on Level 1 return only arrays.

LEVEL 1 skills		
_Skill Syntax	**Result**	**Description**
	_Return	Holds the Result as an array or bundle.
All **Array.math** operations like _min('left', 'right').	Array specified by the dimensions of the first argument (Scalar, Vector, Matrix).	Operates member by member.

LEVEL 2

The result of a previous operation can be used as an argument for the next operation.

```
Bundle.put sBptr, "_Skill", "_*('_Result', '_Result')"

% Returns the square of each entry.
Bundle.get sBptr, "_Result",  mResult[]

% Copies numeric content.
Bundle.put sBptr, "_Skill", "_Copy('ArrayLeft', 'Mem1')"
```

LEVEL 2 skills		
_Skill Syntax	**Result**	**Description**
_Copy('A', 'target')		Copies numeric content. Copy numeric content from a bundle location specified by a bundle key into a different bundle location. This operation works only inside the bundle which is given by the command call. Copying bundles is not intended!

LEVEL 3

The JAMA : The Java Matrix Package (https://math.nist.gov/javanumerics/jama/) is used in the background if needed.

At Level 3, results from type bundle are returned also.

To get access at these values proceed as follows:

```
Bundle.GB sBptr, "_Result",  resPtr % Returns the main Bundle pointer
Bundle.get resPtr, "_EigenvalueMatrix",  EigenvalueMatrix[] % Returns the Matrix
```

or

```
! Copy numeric content from a Bundle into a different main Bundle location.
! Copying bundles is not intended!
Bundle.put sBptr, "_Skill", "_Copy('_Result': '_EigenvalueMatrix', 'Mem1')"
```

<table>
<tr><th colspan="3">LEVEL 3 skills</th></tr>
<tr><th>_Skill Syntax</th><th>Result</th><th>Description</th></tr>
<tr><td colspan="3">Elementary Operations</td></tr>
<tr><td>_Normalize('A')</td><td>Array (Matrix)</td><td>Normalizes
a matrix to make the elements sum to 1.</td></tr>
<tr><td>_ScalarMulti('A', 's')</td><td>Array (Matrix)</td><td>Scalar Multiplication
Multiply a Matrix by a Scalar, A* s</td></tr>
<tr><td>_Times('A', 'B')</td><td>Array (Matrix)</td><td>Matrix Multiplication
Linear algebraic matrix multiplication
Returns Matrix product, A * B</td></tr>
<tr><td>_*('a', 'b')</td><td>Array (Matrix or Vector)</td><td>Dot Product
See level 1</td></tr>
<tr><td>_x3d('a', 'b')</td><td>Array (Matrix or Vector)</td><td>Cross Product
See level 1</td></tr>
<tr><td>_ToggleCtoR('A')</td><td>Array (Matrix)</td><td>ToggleCtoR
Before toggling:
<table><tr><td>\</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>0</td><td>2</td><td>4</td><td>6</td><td>8</td></tr><tr><td>2</td><td>1</td><td>3</td><td>5</td><td>7</td><td>9</td></tr></table>After toggling:
<table><tr><td>\</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>2</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table></td></tr>
<tr><td>_ToggleRtoC('A')</td><td>Array (Matrix)</td><td>ToggleRtoC
Before toggling:
<table><tr><td>\</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>2</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table>After toggling:
<table><tr><td>\</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>0</td><td>2</td><td>4</td><td>6</td><td>8</td></tr><tr><td>2</td><td>1</td><td>3</td><td>5</td><td>7</td><td>9</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="3">LEVEL 3 skills</th></tr>
<tr><th>_Skill Syntax</th><th>Result</th><th>Description</th></tr>
<tr><td>_Transpose('A')</td><td>Array (Matrix)</td><td>Transpose
Before transposing:
<table><tr><td>\</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>0</td><td>2</td><td>4</td><td>6</td><td>8</td></tr><tr><td>2</td><td>1</td><td>3</td><td>5</td><td>7</td><td>9</td></tr></table>
After transposing:
<table><tr><td>\</td><td>1</td><td>2</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>2</td><td>2</td><td>3</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>4</td><td>6</td><td>7</td></tr><tr><td>5</td><td>8</td><td>9</td></tr></table></td></tr>
<tr><td>_UnaryMinus('A')</td><td>Array (Matrix)</td><td>Unary minus
Returns -A</td></tr>
<tr><td colspan="3">Decompositions</td></tr>
<tr><td>_Cholesky('A' {,'B'})</td><td>Bundle [
{ "_L", Array (Matrix) },
{ "_X", Array (Matrix, optional) },
{ "_IsSpd", Array[1] (Scalar) }
]</td><td>Cholesky Decomposition of symmetric, positive definite matrices.

For a symmetric, positive definite matrix A, the Cholesky decomposition is a lower triangular matrix L so that A = L*L'.

B a Matrix with as many rows as A and any number of columns. Returns X so that L*L'*X = B

If the matrix is not symmetric or positive definite, the constructor returns a partial decomposition and sets an internal flag that may be queried by the _IsSpd method.
Returns a scalar array of one member. If 1 it is true. If 0 it is false.</td></tr>
<tr><td>_Lu('A' {,'B'})</td><td>Bundle [
{ "_L", Array (Matrix) },</td><td>LU Decomposition (Gaussian elimination) of rectangular matrices.</td></tr>
</table>

LEVEL 3 skills		
_Skill Syntax	**Result**	**Description**
	{ "_U", Array (Matrix) }, { "_Pivot", Array (Vector) }, { "_X", Array (Matrix, optional) }, { "_Determinant", Array[1] (Scalar) }, { "_IsNonsingular", Array[1] (Scalar) }]	For an m-by-n matrix A with m ≥ n, the LU decomposition is an m-by-n unit lower triangular matrix L, an n-by-n upper triangular matrix U, and a permutation vector piv of length m so that A(piv,:) = L*U. If m < n, then L is m-by-m and U is m-by-n. The LU decomposition with pivoting always exists, even if the matrix is singular, so the constructor will never fail. The primary use of the LU decomposition is in the solution of square systems of simultaneous linear equations. This will fail if isNonsingular() returns 0.0 (false).
_Qr('A' {,'B'})	Bundle [{ "_H", Array (Matrix) }, { "_Q", Array (Matrix) }, { "_R", Array (Matrix) }, { "_X", Array (Matrix, optional) }, { "_IsFullRank", Array[1] (Scalar) }]	**QR Decomposition of rectangular matrices.** For an m-by-n matrix A with m ≥ n, the QR decomposition is an m-by-n orthogonal matrix Q and an n-by-n upper triangular matrix R so that A = Q*R. The QR decomposition always exists, even if the matrix does not have full rank, so the constructor will never fail. The primary use of the QR decomposition is in the least squares solution of nonsquare systems of simultaneous linear equations. This will fail if isFullRank() returns 0.0 (false).
_Svd('A')	Bundle [{ "_S", Array (Matrix) }, { "_U", Array (Matrix) }, { "_V", Array (Matrix) }, { "_SingularValues", Array (Vector) }, { "_SingularValues", Array (Vector) }, { "_Norm2", Array[1] (Scalar) }, { "_Condition", Array[1]	**Singular Value Decomposition of rectangular matrices.** For an m-by-n matrix A with m >= n, the singular value decomposition is an m-by-n orthogonal matrix U, an n-by-n diagonal matrix S, and an n-by-n orthogonal matrix V so that A = U*S*V'.

LEVEL 3 skills		
_Skill Syntax	**Result**	**Description**
	(Scalar) }, { "_Rank", Array[1] (Scalar) }]	The singular values, sigma[k] = S[k][k], are ordered so that sigma[0] ≥ sigma[1] ≥ ... ≥ sigma[n-1]. The singular value decomposition always exists, so the constructor will never fail. The matrix condition number and the effective numerical rank can be computed from this decomposition.
_Eigen('A')	Bundle [{ "_BlockDiagonalEigenvalueMatrix", Array (Matrix) }, { "_EigenvalueMatrix", Array (Matrix) }, { "_ImaginaryPartsEigenvalues", Array (Vector) }, { "_RealPartsEigenvalues", Array (Vector) }]	**Eigenvalue Decomposition of symmetric and square matrices.** If A is symmetric, then A = V*D*V' where the eigenvalue matrix D is diagonal and the eigenvector matrix V is orthogonal. I.e. A = V.times(D.times(V.transpose())) and V.times(V.transpose()) equals the identity matrix. If A is not symmetric, then the eigenvalue matrix D is block diagonal with the real eigenvalues in 1-by-1 blocks and any complex eigenvalues, lambda + i*mu, in 2-by-2 blocks, [lambda, mu; -mu, lambda]. The columns of V represent the eigenvectors in the sense that A*V = V*D, i.e. A.times(V) equals V.times(D). The matrix V may be badly conditioned, or even singular, so the validity of the equation A = V*D*inverse(V) depends upon V.cond().
Equation Solutions		
_Regress('*X*', '*y*'{, 'lambda'}) ***X*** (The first column has to be filled with ones): | \ | 1 | 2 | .. | | 1 | 1 | 156.3 | .. | | 2 | 1 | 158.9 | .. |	Array (Vector)	**Regression** Performs least squares regression using Tikhonov regularization. *X*, the data Matrix (m x n). *y*, the target function values as Array (Vector).

LEVEL 3 skills		
_Skill Syntax	**Result**	**Description**
<table><tr><td>3</td><td>1</td><td>160.8</td><td>..</td></tr><tr><td>4</td><td>1</td><td>179.6</td><td>..</td></tr><tr><td>:</td><td>:</td><td>:</td><td>:.</td></tr></table> y: <table><tr><td>\</td><td>1</td></tr><tr><td>1</td><td>47.1</td></tr><tr><td>2</td><td>46.8</td></tr><tr><td>3</td><td>49.3</td></tr><tr><td>4</td><td>53.2</td></tr><tr><td>:</td><td>:</td></tr></table> See also example below.		If lambda is not specified or < 0 the optimal lambda Scalar will be computed by using generalized cross-validation. If lambda is 0 it works like the conventional $b = (X^T X)^{-1} X^T y$. The result is an Array(Vector) [1] = intersept [2] = $slope_1$: [n] = $slope_{(n-1)}$ y = intersept + $slope_1$ * x_1 + sl...
Derived Quantities		
_Condition('A')	Array[1] (Scalar)	**Condition of a Matrix.**
_Determinant('A')	Array[1] (Scalar)	**Determinant of a Matrix.**
_Identity('rows', 'columns') (Math notation/order!)	Array (Matrix)	**Generates an Identity Matrix.** m = Number of rows. n = Number of columns. Returns an m-by-n matrix with ones on the diagonal and zeros elsewhere.
_Inverse('A')	Array (Matrix)	**Inverse of a Matrix.**
_Norm1('A')	Array[1] (Scalar)	**Norm one** maximum column sum
_Norm2('A')	Array[1] (Scalar)	**Norm two** maximum singular value
_NormF('A')	Array[1] (Scalar)	**Norm Frobenius** sqrt of sum of squares of all elements
_NormInf('A')	Array[1] (Scalar)	**Norm Infinity** maximum row sum
_Rank('A')	Array[1] (Scalar)	**Rank of a Matrix**
_IsSymmetric('A')	Array[1] (Scalar)	**Is Symmetric** Returns 0.0 for false and 1.0 for true.
_Trace('A')	Array[1] (Scalar)	**Sum of the diagonal elements**
_Get('A', 'row', 'column') (Math notation/order!)	Array[1] (Scalar)	**Get the value** at m = Number of row.

LEVEL 3 skills		
_Skill Syntax	**Result**	**Description**
		n = Number of column.?
_SubMatrix('A', 'rows', 'columns') (Math notation/order!)	Array (Matrix)	**SubMatrix** Returns a Sub Matrix specified by rows = {start, end} (Array), columns = {start, end} (Array)?
_RowPacked	Array (Vector)	**RowPacked** Returns a Row Packed Array of the Matrix specified
_ColumnPacked	Array(Vector)	**ColumnPacked**
_Rows('A')	Array[1] (Scalar)	**Rows** Returns the number of rows.
_Columns('A')	Array[1] (Scalar)	**Columns** Returns the number of columns.

LEVEL 4

A bundle entry "_SkillArray" deals with a number of skills.

The pros are:

- You can execute Skill by Skill like _*sin('m1', 'm2'), _*exp(...)….
- Bypassing the interpreter (Speed)
- Programmable

The cons are:

- You should have a good plan
- Should be tested before in small parts
- Exception handling is more difficult

If a bundle entry "_Skill" is also in the bundle, its content will be overwritten by the current operated Array Skill.
If a bundle entry "_Counter" is in the main bundle, it returns the current used Array Skill index.

Example:

```
Fn.def outputArray(n[])
 Array.dims n[], d[]
 Array.length al, d[]
 If al = 2 Then        % Only Matrices
  For r = 1 To d[2]
   For c = 1 To d[1]
    Print Round(n[c, r], 4), "";
   Next
   Print " row", r
  Next
 Else
  For r = 1 To d[1]
   Print Round(n[r], 4), "";
   Print "row: "+ Int$(r)
  Next
```

```
 EndIf
Fn.end

Fn.def getResult(sBptr)
 Bundle.get sBptr, "_Counter", counter
 Print  "_Counter", counter
 Bundle.get sBptr, "_Skill", Skill$
 Print  "_Skill", Skill$
 Bundle.contain sBptr, "_Error", isError
 If isError Then
  Bundle.get sBptr, "_Error", mError$
  Print "_Error !", mError$
 EndIf
 Bundle.contain sBptr, "_Status", isStatus
 If isStatus Then
  Bundle.get sBptr, "_Status", mError$
  Print "_Status", mError$
 ENDIF
 Bundle.contain sBptr, "_Result", isResult
 If isResult Then
  Bundle.type sBptr, "_Result", keyType$
  Print "_Result", keyType$
  If Is_In("N[", keyType$)
   Bundle.get sBptr, "_Result", copyRes[]
   outputArray(copyRes[])
  Else
   Bundle.GB sBptr, "_Result", rBptr
   Debug.on
   Debug.dump.bundle  rBptr
   Debug.off
  EndIf
 EndIf
Fn.end
```

Example:

```
Array.load sT2[], 0, 1, 2, 3, 4, 5, 6, 7, 8  % 9 elements
ReDim 1, sT2[3,3]
outputArray(copyRes[])
Array.mat.toggle sT2[], "_CtoR", sT2[]
outputArray(copyRes[])
Bundle.put sBptr, "A", sT2[]
! Array.load mSkillArray$[], "_Copy('A', 'B')", "_Copy('A', 'F')"
Array.load mSkillArray$[], "_Transpose('A')", "_Transpose('_Result')"
Bundle.put sBptr, "_SkillArray", mSkillArray$[]
Array.mat.skill sBptr, 1
getResult(sBptr)
```

Example of Multiple Linear Regression*:*

```
% KOERPERGROESSE / body height in cm
Array.Load xKgr[], 156.3, 158.9, 160.8, 179.6, 156.6, 165.1, 165.9, 156.7, ~
 167.8, 160.8

% KOERPERGEWICHT / body weight in kg
Array.Load xKge[], 62, 52, 83, 69, 74, 52, 77, 65, 79, 51

% ALTER / age in years
Array.Load xAlt[], 24, 34, 26, 51, 43, 33,  22, 21, 19, 34

% RINGGROESSE / ring size in mm
Array.Load y[], 47.1, 46.8, 49.3, 53.2, 47.7, 49.0, 50.6, 47.1, 51.7, 47.8

Array.length alY, y[]
Dim ones[alY]
Array.Fill ones[], 1
List.create n, mm
List.add.Array mm, ones[]
```

```
List.add.Array mm, xKgr[]
! List.add.Array mm, xKge[] % Comment it out to get a multiple result
! List.add.Array mm, xAlt[] % Comment it out to get a multiple result
List.toArray mm, x[]
Array.length alX, x[]
Array.length alY, y[]
ReDim  1, x[alX/alY,alY]
outputArray(x[])
Bundle.put sBptr, "X", X[]
Bundle.put sBptr, "Y", Y[]
Print "   Conventional regularization b=(X^T * X)^(-1) * X^T * y :"
Array.load mSkillArray$[], ~
 "_Transpose('X')", ~
 "_Copy('_Result', 'Xt')", ~
 "_Times('Xt', 'X')", ~
 "_Inverse('_Result')", ~
 "_Times('_Result', 'Xt')", ~
 "_Times('_Result', 'Y')"
Bundle.put sBptr, "_SkillArray", mSkillArray$[]
Array.mat.skill sBptr, 1
getResult(sBptr)

Print "   Tikhonov regularization :"
Bundle.put sBptr, "lambda", 0 % If lambda 0 the same result as the conventional
Array.load mSkillArray$[], ~
 % "_ToggleCtoR('X')", ~ % If needed
 "_Regress('X', 'Y', 'lambda')"
Bundle.put sBptr, "_SkillArray", mSkillArray$[]
Array.mat.skill sBptr, 1
getResult(sBptr)

Bundle.get sBptr, "_Result", result[]
a = result[1]
b = result[2]

Array.min minX, xKgr[]
Array.max maxX, xKgr[]
leftY = a + b * minX
rightY = a + b * maxX
Print "y = a + b * x"
Print "y = " + Str$(Round(a, 4)) + " + " + Str$(Round(b, 4)) + " * x"
Print "y (Xmin) = " + Str$(Round(leftY, 4))
Print "y (160) = " + Str$(Round(a + b * 160, 4))
Print "y (170) = " + Str$(Round(a + b * 170, 4))
```

Source: https://www.crashkurs-statistik.de/einfache-lineare-regression/#berechnen

and https://www.crashkurs-statistik.de/multiple-lineare-regression/

3.4 Array.math

Syntax: Array.math LeftArray[], RightArray[], <operator_sexp>, ResultArray[]

Operates the left array with the right array by the given operator string <operator_sexp>.

For a command description see the corresponding BASIC! functions.

The dimensions of the result array are equal to the **left** array.

The advantage of this command is the execution speed if compared to **For-Next** loops.

The possible operators are:

_+ _- _* _/ _min _max _atan2 _atan4 _hypot _pow _mod _= _<> _<

_> _<= _>= _shift _bor _band _bxor _huround _: _x3d _x3d1

To round in Half-up mode, use _huround for fast execution.

The operator _: stands also for division, but to prevent infinity or Not-A-Number values, it has a Zero Check option. Instead of returning infinity or NaN, the divider is 10^{-12}. If -0 is detectable, the divider is -10^{-12}.

The cross product of 3d vectors is returned by the _x3d operator.

If the array has more than three entries all three following entries will be interpreted as separate vectors.

The cross product with a vector length of 1 returns _x3d1. It is also zero checked.

_atan4 is special because it returns the radiant for the term y / x in the range [0 … 2*PI].

Example:

```
Array.load left[], 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 % 10 elements
Array.load right[], 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 % 10 elements
Array.math left[], right[], "_+", res[]
Print res[1]  % Returns 0.0
Print res[5]  % Returns 12.0
Array.math left[], right[], "_=", res[]
Print res[3]  % Returns 1.0 for true
```

The following functions are assigned to the members of the **right** array.

(op) can be +, -, *, /, :

_(op)abs	_(op)sgn	_(op)ceil	_(op)floor	_(op)int	_(op)frac	_(op)round
_(op)const_pi	_(op)sin	_(op)sinh	_(op)asin	_(op)cos	_(op)cosh	_(op)acos
_(op)tan	_(op)tanh	_(op)atan	_(op)todegrees		_(op)toradians	
_(op)sqr	_(op)cbrt	_(op)exp	_(op)log	_(op)log10	_(op)const_e	_(op)=0
_(op)<>0	_(op)bnot	_(op)even	_(op)odd	_(op)filter	_(op)2d1vect	_(op)3d1vect

In case of _(op)frac and speed needs see also the description of the function **Frac()**.

The vector length of 1 of 2d vectors will be returned by the _(op)2d1vect operator. If the array has more than two entries, all two following entries will be interpreted as separate vectors. It is also Zero checked.

The vector length of 1 of 3d vectors will be returned by the _(op)3d1vect operator. If the array has more than three entries, all three following entries will be interpreted as separate vectors. It is also Zero checked.

If you have no left array, fill in a dummy with zeros and proceed as follows:

Example:

```
Array.length al, RightArray[]
Dim LeftArray[al]
Array.fill LeftArray[], 0
! Return the content from RightArray[] as absolute values
Array.math LeftArray[], RightArray[], "_+abs", ResultArray[]
! Return boolean values of the RightArray[] as 1.0 for true and 0.0 for false
! if the given value is not 0. Works like an on and off switch.
```

```
Array.math LeftArray[], RightArray[], "_+<>0", ResultArray[]
```

Example of how to scale a 3D vector:

```
Array.load vector[], 1, 1, 4
s = 3
Array.load multi[], s, s, s
! Return the scaled vector. The result is in the vector array on the right side.
Array.math vector[], multi[], "_*", vector[] % Result is 3, 3, 12
```

Example of Simple Linear Regression (regression line):

```
% KOERPERGROESSE
Array.Load x[], 156.3, 158.9, 160.8, 179.6, 156.6, 165.1, 165.9, 156.7, 167.8, ~
 160.8

% RINGGROESSE
Array.Load y[], 47.1, 46.8, 49.3, 53.2, 47.7, 49.0, 50.6, 47.1, 51.7, 47.8

Array.average xAverage, x[] % Get the average of x[]
Array.average yAverage, y[]
Array.length al, x[] % Get the number of array entries
Dim xA[al], yA[al]  % Create new arrays
Array.fill xA[], xAverage % Fill the array for later easy processing
Array.fill yA[], yAverage

Array.math x[], xA[], "_-", xD[] % Operates each x - x Average
Array.math y[], yA[], "_-", yD[]
Array.math xD[], yD[], "_*", yM[] % Multiplies each xD by each yD
Array.math xD[], xD[], "_*", yQ[] % Squares each xD

Array.sum z, yM[] % Get the sum of all array entries
Array.sum n, yQ[]
b = z / n    % slope
a = yAverage - b * xAverage    % intersept

array.min minX, x[]
array.max maxX, x[]
leftY = a + b * minX
rightY = a + b * maxX
Print "xAverage", xAverage
Print "yAverage", yAverage
Print "y = a + b * x"
Print "y = " + Str$(Round(a, 4)) + " + " + Str$(Round(b, 4)) + " * x"
Print "y (Xmin) = " + Str$(Round(leftY, 4))
Print "y (160) = " + Str$(Round(a + b * 160, 4))
Print "y (170) = " + Str$(Round(a + b * 170, 4))
Print "y (Xmax) = " + Str$(Round(rightY, 4))
```

Source:

Taken from https://www.crashkurs-statistik.de/einfache-lineare-regression/#berechnen.

4 Audio Playback Commands

Your program can use the Android Media Player interface for playing music files. This interface is not the most stable part of Android. It sometimes gets confused about what it is doing. This can lead to random "Forced Close" events. While these events are rare, they do occur.

The file types you can play depend on your device and the version of Android it runs. For a current list check the https://developer.android.com/guide/appendix/media-formats.html, also shown in section 1 Supported Media Formats. Here is a partial summary:

Audio File Type	Played by Android Version
AAC AMR MIDI MP3 OGG WAV WMA	all
FLAC	3.1+
AAC-ELD	4.1+
MKV	5.0+

Audio files must be loaded into the Audio File Table (AFT) before they can be played. Each audio file in the AFT has a unique index which is returned by the **Audio.load** command.

4.1 Audio.info

Syntax: Audio.info <aft_nvar>, <bundle_pointer_nvar>

Returns a bundle with the system value keys:

Key	Description
_Album	Name of album.
_Artist	Name of artist.
_Title	Song title.
_BitRate	Audio bit rate.
_Duration	Song time duration.
_LocationPath	
_StreamMetadata	
_CdTrack	CD track number.
_Composer	Name of composer.
_Compilation	
_CaptureFramerate	
_Date	Release date.
_DiscNumber	Disc number.
_StreamTitle	

Streaming data are only available if an Icecast Streaming Server is called.

4.2 Audio.isDone

Syntax: Audio.isDone <lvar>

If the current playing file is still playing then <lvar> will be set to zero otherwise it will be set to one.

This can be used to determine when to start playing the next file in a play list.

```
Audio.play f[x]
Do
 Audio.isDone isdone
 Pause 1000
Until isdone
```

4.3 Audio.length

Syntax: Audio.length <length_nvar>, <aft_nexp>

Note: The **Audio.length** command works in RFO-Basic! but it does not work in OliBasic; use the **Audio.info** command instead.

Returns the total length of the file in the Audio File Table pointed to by <aft_nexp>. The length in milliseconds will be returned in <length_nvar>.

4.4 Audio.load

Syntax: Audio.load <aft_nvar>, <filename_sexp>|<http_stream_sexp>

Loads a music file or internet stream into the Audio File Table. The AFT index is returned in <aft_nvar>. If the file or stream can't be loaded, the <aft_nvar> is set to 0. Your program should test the AFT index to find out if the file or stream was loaded. If the AFT index is 0, you can call the **GetError$()** function to get information about the error. If you use index 0 in another **Audio** command you will get a run-time error.

You can reach outside the "<pref base drive>/ref-basic/data/" by using path fields in the filename. For example, "../../Music/Blue Danube Waltz.mp3" would access "<pref base drive>/Music/Blue Danube Waltz.mp3", or take a look at **File.root**.

Example:

```
File.root dataPath$, "_Music"
fn$ = "file://" + dataPath$ + "/" + "Blue Danube Waltz.mp3"
File.exists ok, fn$
If ok Then
 Audio.load aft, fn$
 If aft Then
  Audio.stop
  Audio.play aft
 EndIf
EndIf
...
Audio.load aft, "http://amp1.cesnet.cz:8000/cro1.ogg"
```

4.5 Audio.loop

Syntax: Audio.loop

When the currently playing file reaches the end of file, the file will restart playing from the beginning of the file. There must be a currently playing file when this command is executed.

4.6 Audio.pause

Syntax: Audio.pause

Pause is like stop except that the next **Audio.play** for this file will resume the play at the point where the play was paused.

4.7 Audio.play

Syntax: Audio.play <aft_nexp> {, <output_nexp>}

Selects the file from the Audio File Table pointed to by <aft_nexp> and begins to play it. There must not be an audio file already playing when this command is executed. If there is a file playing, execute **Audio.stop** first.

The <output_nexp> parameter is used to control the output channels. For the loud speakers or the ear phones set it to 1, and the level is controlled by the Music volume. For both the ear phones and the loud speakers set it to 2,and the level is controlled by the Alarm volume. When this parameter is set, current volume level is saved.

The music stops playing when the program stops running. To simply start a music file playing and keep it playing, keep the program running. This infinite loop will accomplish that:

```
Audio.load ptr, "my_music.mp3"
Audio.play ptr
Do
 Pause 5000
Until 0
```

4.8 Audio.position.current

Syntax: Audio.position.current <nvar>

The current position in milliseconds of the currently playing file will be returned in <nvar>.

4.9 Audio.position.seek

Syntax: Audio.position.seek <nexp>

Moves the playing position of the currently playing file to <nexp> expressed in milliseconds.

4.10 Audio.release

Syntax: Audio.release <aft_nexp>

Releases the resources used by the file in the Audio File Table pointed to by <aft_nexp>. The file must not be currently playing. The specified file will no longer be able to be played.

4.11 Audio.stop

Syntax: Audio.stop {<reset_vol_nexp>},

Audio.stop terminates the currently-playing music file. This command will be ignored is no file is playing. It is best to precede each **Audio.play** command with an **Audio.stop** command.

If <reset_vol_nexp> is set to 1 the volume levels will be reset to the saved value that was saved by the previous **Audio.play** command. If <reset_vol_nexp> is omitted or set to 0, the volume levels will not be reset. Default <reset_vol_nexp> is set to 1.

4.12 Audio.volume

Syntax: Audio.volume <left_nexp>, <right_nexp>{, <outer_nexp>}

Android devices have an inner and an outer volume control. The outer one is controlled by the device's volume control buttons (key 24 and 25). This command changes the inner volume of the left and right stereo channels. There must be a currently playing file when this command is executed.

The values should range between 0.0 (lowest) to 1.0 (highest). The human ear perceives the level of sound changes on a logarithmic scale. The inner volume can only change in the given range of the outer volume control. Optionally, with <outer_nexp>, the outer volume level can be controlled in a range between 0.0 (lowest) to 1.0 (highest). But in this case the logarithmic scale is already taken into account. On **Audio.play** the current outer volume level is saved. On **Audio.stop** or if the program stops, the outer volume level will be reset to the saved levels.

Note: instead of "volume", "sound pressure level" <SPL> is the correct name.

Example:

```
maxVolume = 10 % Wished steps
For vol = 0 To maxVolume
 myVolume = (1 - (LOG(maxVolume- vol) / LOG(maxVolume)))
 !Print mVolume
 Audio.volume myVolume/maxVolume, myVolume/maxVolume, 1
 ! Audio.volume 1, 1, vol/ maxVolume
 Pause 1000
Next
```

5 Audio Record Commands

5.1 Audio.record.buffer

Syntax: Audio.record.buffer <status_nvar>, Array[]

This command is designed **only** for the PCM 16BIT format. See **Audio.record.start**.

No recording to a file is needed. Starting with an empty file name returns also a buffer.

The buffer is created when **Audio.record.buffer** is called the first time. The buffer size depends on the sampling rate, the number of channels and the encoding type.

For example, for a sampling rate of 44100 Hz, 1 channel (Mono) and ~ 25 frames per second (PCM 16BIT format), the array length will be 44100 * 1 / 25 = 1764.

After testing some devices, an accurate value of 25 only seems to apply for a sampling rate of 8000 Hz. A frame-rate-range from 24...25[Hz] was reached for sample rates which are whole-number-multiples of 44100[Hz] (44100,22050,11025) at every tested device. Most, but not all, devices reached the mentioned range of frame rate even with sample rates arbitrary differing from this sequence. The Example can be used to determine buffer sizes for a sequence of sample rates.

The buffer is returned by the numeric array Array[].

Using two audio-channels (**Audio.record.start** <aC_nexp>) instead of one, simply doubles output-buffer-size, the samples are arranged individually paired.

The <status_nvar> variable returns the status. If <status_nvar> is 1, everything is fine. If it is 0, the PCM 16BIT format is not selected or no recording process is running. In the case of -1, the wave file recording has finished due to a file size limit and the array is now 1 in length and has a single value of -1.

If **Audio.record.start** <eC_nexp> is 11 instead of 10 and the file name of **Audio.record.start** is empty, this command waits for each new buffer frame.

See also https://en.wikipedia.org/wiki/Pulse-code_modulation.

Example:

```
Array.load sampRates[], ~
 48000, 44100, 40000, 32000, 24000, 22050, 16000, 11025, 8000
Array.length le, sampRates[]
Print "-------------"
Print "#", "sRate", "bufSize", "frameRate"
For i=1 To le
 Audio.record.start "" , 1, 3, -1, sampRates[i], -1, 1
 Audio.record.buffer id, buf[]
 Audio.record.stop
 Array.length bufSz, buf[]
 Print Int$(i), Int$(sampRates[i]); ",";
 Print Int$(bufsz); ",";
 Print Str$(Round(sampRates[i]/bufSz,2))
Next
Console.save "bufferSizes.txt"
```

5.2 Audio.record.peak

Syntax: Audio.record.peak <level_nvar>

This command returns in <level_nvar> the raw maximum sound pressure level amplitude value. If the PCM 16BIT mode is running, only the last buffer will be evaluated. The **Audio.record.start** command initializes the PEAK process. If no current recording process is available, <level_nvar> returns -1.

5.3 Audio.record.start

Syntax: Audio.record.start <fn_sexp>{{{{{{{{{, <so_nexp>}, <oF_nexp>}, <eC_nexp>}, <sR_nexp>}, <eBR_nexp>}, <aC_nexp>}, <mFS_nexp>}, <lat_nexp>}, <lon_nexp>}

Start audio recording using the microphone as the audio source. The recording will be saved to the specified file. Recording will continue until the A**udio.record.stop** command is issued.

Parameter options:

Argument	Description	Defaults
<fn_sexp>	File Name: If <fn_sexp> is an empty string nothing is saved.	
<so_nexp>	Source: 1* MICROPHONE 5 CAMCORDER 6 VOICE_RECOGNITION 7 VOICE_COMMUNICATION	1
<oF_nexp>	Output Format: 1* THREE_GPP(.g3p) 2 MPEG_4 (.mp4, .m4a) 3 WAVE_16BIT (.wav) and switches autom. to Encoder 10 (PCM_16BIT)	1
<eC_nexp>	Encoder: 1* AMR_NB 2 AMR_WB 3 AAC 4 HE_AAC 5 AAC_ELD 6 Ogg Vorbis (API level 21) 10 PCM_16BIT (autom. switching to Output Format WAVE_16BIT (.wav))	1

Argument	Description	Defaults
	11 PCM_16BIT Waiting for each new buffer frame, if the file name is empty, nothing is saved.	
<sR_nexp>	Sampling Rate: in samples per second. 44100Hz is currently the only rate that is guaranteed to work on all devices, but other rates such as 5.5125, 6.615, 8, 9.6, 11.025, 16, 18.9, 22.05, 27.42857, 32, 33.075, 37.8, 44.1, 48 kHz. may work on some devices.	44100
<eBR_nexp>	Encoding Bit Rate: in bits per second Ignored by PCM_16BIT	96000
<aC_nexp>	Audio Channels: 1 or 2* For one channel you get a mono output in a stereo file if a PCM 16 BIT encoder is not chosen!	2
<mFS_nexp>	Maximum File Size: in bytes, but the real size is more less! If the maximum file size is reached, the recording stops sooner or later!	-1
<lat_nexp>	Latitude: -90 to 90 degree Ignored by PCM_16BIT	
<lon_nexp>	Longitude: -180 to 180 degree Ignored by PCM_16BIT	

Note: Some results are device, sampling rate, encoding bit rate and/or codec dependent.

Example:

```
Rem Start of BASIC! Program audio_recording.bas
Device dbp
Bundle.get dbp, "OS", myOs$
Print myOs$
Debug.on
filename$ = "NewSound.g3p"
filename$ = "NewSound.m4a"
! Command OPTIONS:
so = 1 %Source 0 Default; 1* Mic; 5 CAMCORDER; 6 VOICE_RECOGNITION;~
 ! (7 VOICE_COMMUNICATION)
oF = 2 %OutputFormat 1* THREE_GPP(.g3p); 2 MPEG_4 (.mp4, .m4a)
eC = 3 %Encoder 1* AMR_NB; (2 AMR_WB; 3 AAC); (4 HE_AAC; 5 AAC_ELD)
sR = 48000 %(SamplingRate in samples per second) 44100*
eBR = 80000 %(EncodingBitRate in bits per second) 96000*
aC = 2 %(AudioChannels 1 or 2*) For one channel you get a mono output in a
stereo file!
mFS = 280000 %MaxFileSize in bytes, but the real size is more less! -1*
! If the max. file size is reached, the recording stops sooner or later!
lat = 50 %(Latitude -90 to 90 degree)
lon = 150 %(Longitude -180 to 180 degree)

Audio.record.start filename$,so,oF,eC,sR,eBR,aC,mFS,lat,lon
For i = 1 To 20
 Pause 1000
 ! With this PEAK command you get the max. amplitude between to calls.
```

```
    ! The START command init the PEAK process for the first time.
    ! If no current recording process available, PEAK returns -1.
    Audio.record.peak m
    File.size FS, filename$
    Print "Peak "; m; " File size "; fS
  Next

  File.size fS, filename$
  PRINT " Sound file size in byte: ";fS
  Audio.record.stop
  Audio.load ptr, filename$
  Audio.play ptr
```

See also: https://developer.android.com/guide/topics/media/media-formats.html.

5.4 Audio.record.stop

Syntax: Audio.record.stop

Stops the previously started audio recording.

6 Background Commands and Functions

A running program continues to run when the HOME key is tapped. This is called running in the Background. When not in the Background mode, the program is in the Foreground mode. The program exits the Background mode and enters the Foreground mode when the BASIC! icon on the home screen is tapped.

6.1 Background

Syntax: Background()

This function returns one of the following values:

Value Returned	Description
0	The program is not running in the background.
1	The program is running in the background.
3	Display screen is off.
4	Device is locked.
5	Display screen is on and the device is locked.

Sometimes you will want to know if the program is running in the Background. One reason for this might be to stop music playing while in the Background mode. If you want to be able to detect Background mode while Graphics is open, you must not call **Gr.render** while in the Background mode. Doing so will cause the program to stop running until the Foreground mode is re-entered. Use the following code line for all **Gr.render** commands:

```
If !Background() Then Gr.render
```

6.2 Background.resume

Syntax: Background.resume

This statement should be placed at the end of the interrupt handler at **OnBackground:**.

6.3 Home

Syntax: Home

The **Home** command does exactly what tapping the HOME key would do. The Home Screen is displayed while the program continues to run in the background.

Running on Android 4.2.1 with 512KB RAM, your program might not continue under certain circumstances. Starting with Android 5, background tasks are more reliable.

The **Home** command does the opposite of **Console.front** and **Gr.front**. Also see the suggested **ReorderToFront()** function in the **App.SAR** examples.

6.4 OnBackground:

Syntax: OnBackground:

Label for an interrupt handler which traps changes in the Background/Foreground state. The **Background()** function can be used to determine the new state. When done, execute the **Background.resume** statement to resume the interrupted program.

6.5 WakeLock

Syntax: WakeLock <code_nexp>{, <flags_nexp>}

The **WakeLock** command modifies the system screen timeout function. The Code parameter <code_nexp> may be one of five values. Values 1 through 4 modify the screen timeout in various ways. Code value 5 releases the WakeLock and restores the system screen timeout function.

The following table lists the Code parameters.

Code	WakeLock Type	CPU	Screen	Keyboard Light	More Information
1	Partial WakeLock	On*	Off	Off	http://developer.android.com/reference/android/os/PowerManager.html#PARTIAL_WAKE_LOCK
2	Screen Dim	On	Dim	Off	http://developer.android.com/reference/android/os/PowerManager.html#SCREEN_DIM_WAKE_LOCK
3	Screen Bright	On	Bright	Off	http://developer.android.com/reference/android/os/PowerManager.html#SCREEN_BRIGHT_WAKE_LOCK
4	Full WakeLock	On	Bright	Bright	http://developer.android.com/reference/android/os/PowerManager.html#FULL_WAKE_LOCK
5	No WakeLock	Off	Off	Off	

** If you hold a partial WakeLock, the CPU will continue to run, regardless of any timers and even after the user taps the power button. In all other WakeLocks, the CPU will run, but the user can still put the device to sleep using the power button.*

You can use the optional Flags parameter <flags_nexp> to change the screen behavior, as shown in the table below. If the WakeLock Type is Partial WakeLock (Code 1), the Flags parameter is ignored.

Value	Flag(s) Set	Meaning	More Information
1	Acquire causes wakeup	Turn screen on when wakelock is acquired.	http://developer.android.com/reference/android/os/PowerManager.html#ACQUIRE_CAUSES_WAKEUP
2	On after release	Reset screen timeout when wakelock is released.	http://developer.android.com/reference/android/os/PowerManager.html#ON_AFTER_RELEASE
3	Both		
Other	Neither		

Use the WakeLock only when you really need it. Acquiring a WakeLock increases power usage and decreases battery life. The WakeLock is always released when the program stops running.

One of the common uses for **WakeLock** would be in a music player that needs to keep the music playing after the system screen timeout interval. Implementing this requires that your program be kept running. One way to do this is to put it in an infinite loop:

```
Audio.load n,"B5b.mp3"
Audio.play n
WakeLock 1
Do
 Pause 30000
```

```
Until 0
```

The screen will turn off when the system screen timeout expires but the music will continue to play.

6.6 WiFiLock

Syntax: WiFiLock <code_nexp>

The **WiFiLock** command allows your program to keep the WiFi connection awake when a time-out would normally turn it off. The <code_nexp> may be one of four values. Values 1 through 3 acquire a WiFiLock, changing the way WiFi behaves when the screen turns off. Code value 4 releases the WiFiLock and restores the normal timeout behavior.

Code	WiFiLock Type	WiFi Operation When Screen is Off	More Information
1	Scan Only	WiFi kept active, but only operation is initiating scan and reporting scan results.	http://developer.android.com/reference/android/net/wifi/WifiManager.html#WIFI_MODE_SCAN_ONLY
2	Full	WiFi behaves normally.	http://developer.android.com/reference/android/net/wifi/WifiManager.html#WIFI_MODE_FULL
3	Full High Performance	WiFi operates at high performance with minimum packet loss and low latency*.	http://developer.android.com/reference/android/net/wifi/WifiManager.html#WIFI_MODE_FULL_HIGH_PERF
4	No WiFiLock	WiFi turns off (may be settable on some devices).	

*Full Hi-Perf mode is not available on all Android devices. Devices running Android 3.0.x or earlier, and devices without the necessary hardware, will run in Full mode instead.

Use **WiFiLock** only when you really need it. Acquiring a WiFiLock increases power usage and decreases battery life. The WiFiLock is always released when the program stops running.

7 Bluetooth Commands

Bluetooth is implemented in a manner which allows the transfer of data bytes between an Android device and some other device (which may or may not be another Android device).

Before attempting to execute any Bluetooth commands, you should use the Android "Settings" Application to enable Bluetooth and pair with any device(s) with which you plan to communicate.

When Bluetooth is opened using the **Bt.open** command, the device goes into the Listen Mode. While in this mode it waits for a device to attempt to connect.

For an active attempt to make a Bluetooth connection, you can use the Connect Mode by successfully executing the **Bt.connect** command. Upon executing the **Bt.connect** command the person running the program is given a list of paired Bluetooth devices and asked. When the user selects a device, your program attempts to connect to it.

You should monitor the state of the Bluetooth using the **Bt.status** command. This command will report states of Listening, Connecting and Connected. Once you receive a "Connected" report, you can proceed to read bytes and write bytes to the connected device.

You can write bytes to a connected device using the **Bt.write** command.

Data is read from the connected device using the **Bt.read.bytes** command; however, before executing **Bt.read.bytes**, you need to find out if there is data to be read. You do this using the **Bt.read.ready** command.

Once connected, you should continue to monitor the status (using **Bt.status**) to ensure that the connected device remains connected.

When you are done with a particular connection or with Bluetooth in general, execute **Bt.close**.

The sample program, f35_bluetooth, is a working example of Bluetooth using two Android devices in a "chat" type application.

7.1 Bt.close

Syntax: Bt.close

Closes any previously opened Bluetooth connection. Bluetooth will automatically be closed when the program execution ends.

7.2 Bt.connect

Syntax: Bt.connect {0|1}

Commands connection to a particular device. Executing this command will cause a list of paired devices to be displayed. When one of these devices is selected the **Bt.status** will become "Connecting" until the device has connected.

The optional parameter determines if BT will seek a secure or insecure connection. If no parameter is given or if the parameter is 1, then a secure connection will be requested. Otherwise, an insecure connection will be requested.

7.3 Bt.connect.address

Syntax: Bt.connect.address <address_svar>{, 0|1}

Commands connection to a particular device by the given Bluetooth address <address_svar>. If you

use a string from **Bt.name** or **Bt.paired**, the address is extracted automatically. Executing this command will cause a list of paired devices to be displayed. **Bt.status** will become "Connecting" until the device has connected.

The optional parameter determines if BT will seek a secure or insecure connection. If no parameter is given or if the parameter is 1, then a secure connection will be requested. Otherwise, an insecure connection will be requested.

If an error occurs <address_svar> returns a string starting with "Connect error:"

Example:

```
btAddress$ = "29:BF:26:84:06:1B"
Bt.connaect.address btAddress$
If Is_In("Connect error:", btAddress$) THEN Bt.connect
```

7.4 Bt.device.name

Syntax: Bt.device.name <svar>

Returns the name of the connected device in the string variable. A run-time error will be generated if no device (Status <> 3) is connected.

7.5 Bt.disconnect

Syntax: Bt.disconnect

Disconnects from the connected Bluetooth device and goes into the Listen status. This avoids having to use **Bt.close** + **Bt.open** to disconnect and wait for a new connection.

7.6 Bt.onReadReady.resume

Syntax: Bt.onReadReady.resume

This statement should be placed at the end of the interrupt handler at **OnBtReadReady**.

7.7 Bt.onStatus.resume

Syntax: Bt.onStatus.resume

This statement should be placed at the end of the interrupt handler at **OnBtStatus**.

7.8 Bt.open

Syntax: Bt.open {{{0|1}, <delimiter_nexp>}, <del_nexp>}

Opens Bluetooth in Listen Mode. If you do not have Bluetooth enabled (using the Android Settings Application) then the person running the program will be asked whether Bluetooth should be enabled. After **Bt.open** is successfully executed, the code will listen for a device that wants to connect.

The optional parameter determines if BT will listen for a secure or insecure connection. If no parameter is given or if the parameter is 1, then a secure connection request will be listened for. Otherwise, an insecure connection will be listened for. It is not possible to listen for either a secure or insecure connection with one **Bt.open** command because the Android API requires declaring a specific secure/insecure open.

If **Bt.open** is used in graphics mode (after **Gr.open**), you will need to insert a **Pause 500** statement after the **Bt.open** statement.

In most cases, the counterpart of a Bluetooth connection uses a line feed at the end of a message.

The bytes of a message will be stored in memory until the character code of <delimiter_nexp> (greater than -1) is reached. Default is 10, which is equal to a line feed / newline.

After receiving the delimiter, the content of the memory will be returned by **Bt.read.bytes** or **Bt.utf_8.read.bytes**. Should the delimiter also be deleted, <del_nexp> has to be greater than 0. Default is 1.

Iif you are dealing with microcontrollers, and you try to handle long messages (> 128 bytes), the sender should send two line feeds at the end of a message. This is because the Serial Port Profile (SPP) maximum payload capacity is device-dependent from 128 to 1024 bytes. Note that UTF-8 characters can have one or two bytes. It seems that connections between Android devices split the messages larger than about 400 bytes as needed.

The maximum bit rate of a serial service for a Bluetooth connection should be 115200 bits per second. See also:

https://electronics.stackexchange.com/questions/203763/bluetooth-module-throughput-uart-baud-rate-how-fast-is-it.

To get the old command characteristic, set <delimiter_nexp> to -1.

7.9 Bt.paired

Syntax: Bt.paired <list_nexp>

Returns a string list of paired Bluetooth devices. Name and address, delimited by a linefeed "\n", will be returned for each available entry. It may take several successful connections before Android accepts a device as paired.

7.10 Bt.read.bytes

Syntax: Bt.read.bytes <svar>

The next available message is placed into the specified string variable. If there is no message then the string variable will be returned with an empty string ("").

Each message byte is placed in one character of the string; the upper byte of each character is 0. This is similar to **Byte.read.buffer**, which reads binary data from a file into a buffer string.

If you need the full Unicode character set use **Bt.utf_8.read.bytes**.

7.11 Bt.read.ready

Syntax: Bt.read.ready <nvar>

Reports in the numeric variable the number of messages ready to be read. If the value is greater than zero then the messages should be read until the queue is empty.

7.12 Bt.reconnect

Syntax: Bt.reconnect

This command will attempt to reconnect to a device that was previously connected (during this Run) with **Bt.connect** or a prior **Bt.reconnect**. The command cannot be used to reconnect to a device that was connected following a **Bt.open** or **Bt.disconnect** command (i.e. from the **Listening** status).

You should monitor the Bluetooth status for **Connected** (3) after executing **Bt.reconnect**.

7.13 Bt.set.UUID

Syntax: Bt.set.UUID <sexp>

A Universally Unique Identifier (UUID) is a standardized 128-bit format for a string ID used to uniquely identify information. The point of a UUID is that it's big enough that you can select any random 128-bit number and it won't clash with any other number selected similarly. In this case, it's used to uniquely identify your application's Bluetooth service. To get a UUID to use with your application, you can use one of the many random UUID generators on the web.

Many devices have common UUIDs for their particular application. The default BASIC! UUID is the standard Serial Port Profile (SPP) UUID: "00001101-0000-1000-8000-00805F9B34FB".

You can change the default UUID using this command.

7.14 Bt.status

Syntax: Bt.status {{<connect_var>}{, <name_svar>}{, <address_svar>}}

Gets the current Bluetooth status and places the information in the return variables. The available data are the current connection status (in <connect_var>), and the friendly name and MAC address of your Bluetooth hardware (in <name_svar> and <address_svar>).

All parameters are optional; use commas to indicate omitted parameters.

If the connection status variable <connect_var> is present, it may be either a numeric variable or a string variable. The table shows the possible return values of each type:

Numeric Value	String Value	Meaning
-1	Not enabled	Bluetooth not enabled
0	Idle	Nothing going on
1	Listening	Listening for connection
2	Connecting	Connecting to another device
3	Connected	Connected to another device
4	Lost	Connection lost and try again
5	Failed	Getting a connection failed and try again

If the device name string variable <name_svar> is present, it is set to the friendly device name. If your device has no Bluetooth radio, the string will be empty.

If the address string variable <address_svar> is present, it is set to the MAC address of your Bluetooth hardware, represented as a string of six hex numbers separated by colons: "00:11:22:AA:BB:CC".

7.15 Bt.utf_8.read.bytes

Syntax: Bt.utf_8.read.bytes

See Bt.read.bytes

7.16 Bt.utf_8.write

Syntax: Bt.utf_8.write {<exp> {,|;}} ...

See **Bt.write**

7.17 Bt.write

Syntax: Bt.write {<exp> {,|;}} ...

Writes data to the Bluetooth connection.

If the comma (,) separator is used, then a comma will be printed between the values of the expressions.

If the semicolon (;) separator is used, then nothing will separate the values of the expressions.

If the semicolon is at the end of the line, the output will be transmitted immediately, with no newline character(s) added.

The parameters are the same as the **Print** parameters. This command is essentially a **Print** to the Bluetooth connection, with two differences:

- Only one byte is transmitted for each character; the upper byte is discarded. Binary data and ASCII text are sent correctly, but Unicode characters may not be. If you need the full Unicode character set, use the **Bt.utf_8.write** syntax.
- A line that ends with a semicolon is sent immediately, with no newline character(s) added.

This command with no parameters sends a newline character to the Bluetooth connection.

7.18 OnBtReadReady:

Syntax: OnBtReadReady:

Label for an interrupt handler which traps the arrival of a message received on the Bluetooth channel. If a Bluetooth message is ready (**Bt.read.ready** would return a non-zero value) your ptogram executes the statements after the **OnBtReadReady:** label, where you can read and handle the message. When done, execute the **Bt.onReadReady.Resume** statement to resume the interrupted program.

7.19 OnBtStatus:

Syntax: OnBtStatus:

Label for an interrupt handler which traps a change in the status of the bluetooth connection. When done, execute the **Bt.onStatus.resume** statement to resume the interrupted program.

8 Bluetooth Low Energy (BLE) Commands

The following has been extracted from: https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/.

Bluetooth Low Energy, BLE for short, is a power-conserving variant of Bluetooth. BLE's primary application is short distance transmission of small amounts of data (low bandwidth). Unlike Bluetooth that is always on, BLE remains in sleep mode constantly except for when a connection is initiated.

This makes it consume very low power. BLE consumes approximately 100x less power than Bluetooth (depending on the use case).

Additionally, BLE supports not only point-to-point communication, but also broadcast mode, and mesh network.

Also see: https://www.tapatalk.com/groups/rfobasic/get-connected-with-an-esp32-over-ble-an-option-for-t6489.html#p48717

The following example, contributed by forum member Tino (tino1003870), shows how to communicate with a Sensirion temperature and humidity sensor SHT41 via BLE commands:

```
Fn.def float(data$)
 !input data: chr-string aka chr(41)chr(ad)chr(8/)chr(1f)
 !result: 21.690000534057617
 n1 = Ascii(data$,1)
 n2 = Ascii(data$,2)
 n3 = Ascii(data$,3)
 n4 = Ascii(data$,4)
 !combine the bytes to one decimal
 n = Shift(n4, -24)
 n = n + Shift(n3, -16)
 n = n + Shift(n2, -8)
 n = n + n1
 !extract sign, exponent and mantissa
 sign     =       Band(n, Bin("10000000000000000000000000000000"))
 exponent = Shift(Band(n, Bin("01111111100000000000000000000000")), 23) - 127
 number   =       Band(n, Bin("00000000011111111111111111111111"))
 !if the sign bit is 1, we have a negavtive number, else it is positive
 If sign = 0
  sign = 1
 Else
  sign = -1
 EndIf
 !convert mantissa to binary string
 zm$ = Bin$(number)
 !fill up binary string up to 23 digits
 If Len(zm$) < 24 then
  Do
   zm$ = "0" + zm$
  Until Len(zm$) = 23
 EndIf
 !mantissa is the float value after the decimal point between 1 and 2, so
 !we start adding from 1 in the target number z
 z = 1
 For i = 1 To 23
  If Mid$(zm$, i, 1) = "1"
   !if bit is set, add 0.25, 0.25, 0.125, ...
   z = z + 1/(2^i)
  EndIf
 Next i
 !now we have the sign, the number and the exponent ... combine it
 z = Sign * z * (2 ^ exponent)
```

```
 !done ... z holds the IEE754 floating number
 Fn.rtn z
Fn.end

!---main---

! --- sht31
datatemp$ = "00002235-B38D-4985-720E-0F993A68EE41"
datahumi$ = "00001235-B38D-4985-720E-0F993A68EE41"
! my dev: "FE:4A:56:D5:4B:14"

Cls
rssi = 0
n$ = ""
Ble.open
Ble.scan 1

If n$ <> "" Then quit
scan:
Cls
Print "Scanning..."
Ble.devices dl$[]
Array.length len, dl$[]
For i = 1 To len
 Ble.rssi dl$[i], rssi
 Ble.name dl$[i], n$
 If dl$[i] <> "" Then
  s$ = " " + n$
  Print dl$[i] ; " rssi=" ; rssi;" ";n$
  !sht31
  !if is_in(dl$[i],"FE:4A:56:D5")>0
  !sht40
  !if is_in(s$,"D7:6E:5E:90")>0
  If Mid$(s$, 1, 6) = " SHT40"
   Print "my device found"
   dev$ = dl$[i]
   GoTo connectme
  EndIf
 EndIf
Next i
Pause 1000
GoTo scan

OnConsoleTouch:
Console.line.touched line
dev$ = dl$[line - 1]
Ble.scan 0
Print "chosen device: ";dev$

connectme:
Ble.scan 0
Do
 Ble.status st$
 If st$ <> last$ Then
  last$ = st$
  Print st$
 EndIf
 If st$ = "Disconnected" Then
  Ble.connect dev$
 EndIf
 Pause 200
Until st$ = "Connected"

Pause 500
Ble.notify datatemp$, 1
Ble.notify datahumi$, 1

tm = 0
```

```
loop:
Pause 1000
Ble.read data$
If data$ <> "" Then
 Print ". ";
 z = Float(data$)
 Print "T= ";z;" *C"
EndIf
GoTo loop
```

```
Disconnected
Connecting
Discovering
Connected
. T= 11.803234100341797 *C
. T= 11.749828338623047 *C
. T= 11.440071105957031 *C
. T= 11.258487701416016 *C
. T= 11.11962890625 *C
. T= 11.106277465820312 *C
. T= 11.04486083984375 *C
. T= 11.031509399414062 *C
```

This is a screenshot of a SHT4x SmartGadget connected via BLE, on a Samsung Tablet using OliBasic:

8.1 Ble.characteristics

Syntax: Ble.characteristics <service_sexp>, Array$[]

Returns characteristics of a connected device for a given service.

8.2 Ble.close

Syntax: Ble.close

Closes the BLE system.

8.3 Ble.connect

Syntax: Ble.connect <device_sexp>

Connects to a device that was previously found.

8.4 Ble.devices

Syntax: Ble.devices Array$[]

Returns nearby devices found, in the form of addresses like XX:XX:XX:XX:XX:XX.

8.5 Ble.disconnect

Syntax: Ble.disconnect

Disconnects from the device.

8.6 Ble.notify

Syntax: Ble.notify <char_sexp>, <flag_nexp>

Enables(1) / disables(0) notifications for a given characteristic.

8.7 Ble.open

Syntax: Ble.open

Opens the BLE system.

8.8 Ble.read

Syntax: Ble.read <data_svar>

Read data from a characteristic (empty if not yet received).

8.9 Ble.read.request

Syntax: Ble.read.request <char_sexp>

Request a read for given characteristic.

8.10 Ble.rssi

Syntax: Ble.rssi <device_sexp>, <rssi_nvar>

Returns the RSSI (Received Signal Strength Indicator) in dBm for a given device address. For more information, see:

https://developer.radiusnetworks.com/2014/12/04/fundamentals-of-beacon-ranging.html

8.11 Ble.scan

Syntax: Ble.scan <flag_nexp>

Enables(1) / disables(0) BLE scanning.

8.12 Ble.scan.record

Syntax: Ble.scan.record <device_sexp>, <record_svar>

Returns the raw scan record of a BLE device.

8.13 Ble.services

Syntax: Ble.services Array$[]

Returns services on a connected device.

8.14 Ble.status

Syntax: Ble.status <device_sexp>, <name_svar>

Returns the device name for a given device address.

OR

Syntax: Ble.status <status_svar>

Gets the connection status ("Disconnected", "Scanning", "Connecting", "Discovering" "Connected").

8.15 Ble.write

Syntax: Ble.write <char_sexp>, <data_sexp>

Writes data to a device.

9 Broadcast Commands

Broadcasts are messages from the system or other applications (activities). You can send and receive these messages with broadcasts. Before you are able to receive broadcasts, you have to initialize a so called Broadcast Receiver. The Broadcast Message is called an Intent. The Intent will be received by any application that has the right Intent Filter.

If you send messages between instances of your program (different package IDs are needed) at runtime, maybe use this action addresses:

- Prog 1 use for sending "com.rfo.basicFellow.broadcast.SEND1" and receiving "com.rfo.basicOli.broadcast.SEND2"
- Prog 2 use for sending "com.rfo.basicOli.broadcast.SEND2" and receiving "com.rfo.basicFellow.broadcast.SEND1"

The use of **App.broadcast** is not recommended, but **App.SAR** can be used to send a Broadcast with much more options.

Example:

```
List.create S, commandListPointer
List.add commandListPointer ~
 "new Intent("+Chr$(34)+"com.rfo.basic.broadcast.SEND"+Chr$(34)+");" ~
 ! → );" ~ ← is important
 "EOCL"
Bundle.PL appVarPointer,"_CommandList",commandListPointer
Bundle.put appVarPointer,"_Broadcast",""
App.SAR appVarPointer
```

See Also: **TGet**

9.1 Broadcast.init

Syntax: Broadcast.init <action_sexp> | Array$[]

Initializes a Broadcast Receiver

9.2 Broadcast.in

Syntax: Broadcast.in <recAction_sexp>, <retData_svar>, <retBundleIndex_nvar>

Receives a Broadcast message (Intent) according to the <recAction_sexp> action address. With <retData_svar> you get the Data Extra string from the Intent. The <retBundleIndex_nvar> returns a bundle. If no data is broadcasting, then <retData_svar> returns "" and <retBundleIndex_nvar> returns an empty bundle.

You have also the option to detect system broadcasts, but in some cases you need also the right permissions. For example, for the broadcast filter "android.net.conn.INET_CONDITION_ACTION" you need the new permission ACCESS_NETWORK_STATE. Today the two available compilers are still not capable for broadcast permission autodetection. That is your job. You can find more information at:

https://android.googlesource.com/platform/frameworks/base/+/master/core/res/AndroidManifest.xml

9.3 Broadcast.close

Syntax: Broadcast.close

Closes the Broadcast Receiver

9.4 Broadcast.bundle

Syntax: Broadcast.bundle <sndAction_sexp>, <key_sexp>, <bundle_ptr_nvar>{, <ordered_nexp>}

Sends a Broadcast message as a Bundle with the action address sndAction and the key.

An ordered Broadcast is send, if <ordered_nexp> is > 0. Default is 0.

9.5 Broadcast.resume

Syntax: Broadcast.resume

This statement should be placed at the end of the interrupt handler at **OnBroadcast:**.

9.6 Broadcast.string

Syntax: Broadcast.string <sndAction_sexp>, <key_sexp>, <msg_sexp>{, <ordered_nexp>}

Sends a Broadcast message as a String with the action address sndAction and the key.

An ordered Broadcast is send, if <ordered_nexp> is > 0. Default is 0.

9.7 KB.send.keyEvent

Syntax: KB.send.keyEvent <tapType_nexp>, <actionType_sexp>, <keyCode_nexp>

Sends a key event internally as a Broadcast message to the environment.

Supported tap types <tapType_nexp>	
0	Key down
1	Key up
4	Key down and up

Action type group <actionType_sexp>	
Value	**Intent action (informative)**
_Global (no permission)	android.intent.action.GLOBAL_BUTTON
_Call	android.intent.action.CALL_BUTTON
_Media	android.intent.action.MEDIA_BUTTON
_Camera	android.intent.action.CAMERA_BUTTON

For key events <keyCode_nexp> see:

https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/view/KeyEvent.java

Mostly apps ignore these events. Try some media players like VLC, Gplayer or TotalCommander.

Example:

```
upAndDown = 4
```

```
type$ = "_media"
keyCode = 127      % Media PAUSE button
Kb.send.keyEvent upAndDown, type$, keyCode
Pause 3000
Kb.send.keyEvent upAndDown, type$, keyCode
```

9.8 OnBroadcast:

Syntax: OnBroadcast:

Label for an interrupt handler which traps a received broadcast. When done, execute the **Broadcast.resume**. statement to resume the interrupted program.

10 Bundle Commands

A Bundle is a group of values collected together into a single object. A bundle object may contain any number of string and numeric values. Also allowed are arrays, lists, stacks, bundles, booleans, drawables and bitmaps.

There is no fixed limit on the size or number of bundles. You are limited only by the memory of your device.

The values are set and accessed by keys. A key is a string that identifies the value. For example, a bundle might contain a person's first name and last name. The keys for accessing those name strings could be "first_name" and "last_name". An age numeric value could also be placed in the Bundle using an "age" key.

A new, empty bundle is created by using the **Bundle.create** command. The command returns a pointer to the empty bundle. Because the bundle is represented by a pointer, bundles can be placed in lists and arrays. Bundles can also be contained in other bundles. This means that the combination of lists and bundles can be used to create arbitrarily complex data structures.

After a bundle is created, keys and values can be added to the bundle using the **Bundle.put** command. Those values can be retrieved using the keys in the **Bundle.get** command. There are other bundle commands to facilitate the use of bundles.

Bundle Auto-Create

Every bundle command except **Bundle.create** has a parameter, the <pointer_nexp>, which can point to a bundle. If the expression value points to a bundle, the existing bundle is used. If it does not, and the expression consists only of a single numeric variable, then a new, empty bundle is created, and the variable value is set to point to the new bundle.

That may seem complex, but it isn't, really. If there is a bundle, use it. If there is not, try to create a new one – but BASIC! can't create a new bundle if you don't give it a variable name. BASIC! uses the variable to tell you how to find the new bundle.

```
% Put a value in the bundle pointed to by b
Bundle.put b,"key1", 1.2

% Put a value in the 10th bundle created
Bundle.put 10, key2$, value2

% Remove a key/value pair from a bundle pointed to by c + d
Bundle.remove c + d, key$[3],
```

In the first example, if the value of **b** points to a bundle, the **Bundle.put** puts **"key1"** and the value **1.2** into that bundle. If **b** is a new variable, its value is 0.0, so it does not point to a bundle. In that case, the **Bundle.put** creates a new bundle, puts **"key1"** and the value **1.2** into the new bundle, and sets **b** to point to the new bundle.

In the second example, if there are at least ten bundles, then the **Bundle.put** tries to put the key named in the variable **key2$** and the value of the variable **value2** into bundle 10. If there is no bundle 10, then the command does nothing. It can't create a new variable because you did not provide a variable to return the bundle pointer.

In the third example, the bundle pointer is the value of the expression **c + d**. If there is no such bundle, the command does nothing. To create a new bundle, the bundle pointer expression must be a single numeric variable.

Bundles in Graphics

The bundle commands that deal with graphics are described in the Graphics chapter. The commands are: Gr.bitmap.get, Gr.bitmap.put, Gr.drawable.get, Gr.drawable.put.

10.1 Bundle.clear

Syntax: Bundle.clear <pointer_nexp>

The bundle pointed to by <pointer_nexp> will be cleared of all tags. It will become an empty bundle. If the bundle does not exist, a new one may be created.

10.2 Bundle.contain

Syntax: Bundle.contain <pointer_nexp>, <key_sexp> , <contains_nvar> {, ...}

If the key specified in the key string expression is contained in the bundle's keys then the "contains" numeric variable will be returned with a non-zero value. The value returned will be zero if the key is not in the bundle. If the bundle does not exist, a new one may be created. The <key_sexp>, <contains_nvar> parameters may be repeated multiple times for different keys.

10.3 Bundle.copy

Syntax: Bundle.copy <sourcePointer_nexp>, <destinationPointer_nexp>

Copies the source bundle to the destination Bundle.

10.4 Bundle.create

Syntax: Bundle.create <pointer_nvar>

A new, empty bundle is created. The bundle pointer is returned in <pointer_nvar>.

10.5 Bundle.GB

Syntax: Bundle.GB <pointer_nexp>, <key_sexp>, <bundle_ptr_nvar>

Short for Bundle.Get.Bundle.

Places the bundle specified by the key string expression into the pointer <bundle_ptr_nvar> specified bundle. If the bundle specified by <pointer_nexp> does not exist or does not contain the requested key, the command generates a run-time error.

If the bundle specified by <**bundle_ptr**_nvar> does not exist a new one will be created.

Example:

```
BUNDLE.GB bptr,"bundleContainer", bundlePtr
```

10.6 Bundle.get

Syntax: Bundle.get <pointer_nexp>, <key_sexp>, <value_nexp> | Array[] | <value_sexp> | Array$[] { ..., <key_sexp>, <value_nexp> | Array[] | <value_sexp> | Array$[] ...}

Places the value specified by the key string expression into the specified numeric or string variable. The type (string or numeric) of the destination variable must match the type stored with the key. If the bundle does not exist or does not contain the requested key, the command generates a run-time error.

Example:

```
Bundle.get bptr,"first_name", first_name$
Bundle.get bptr,"age", age
```

Places the value specified by the key string expression into the specified numeric or string variable. The type (array, string or numeric) of the destination variable must match the type stored with the key. The exception is an Object as an incoming data type. This Object will be converted as much as possible into a string value.

If a number is stored as a String, **Bundle.get** may in some cases be able to return the value as a number.

Example:

```
Bundle.get bptr, "first_name", first_name$, "last_name", last_name$
Bundle.get bptr, "age", age
Bundle.get bptr, "professions", professions$[]
```

10.7 Bundle.GJ

Syntax: Bundle.GJ <pointer_nexp>, <json_sexp>{, <spaces_nexp>}

Short for Bundle.Get.JSON.

Gets the content of a bundle into a JSON string. Supported data types are Strings, Doubles, Booleans as single value or Array. If the optional <spaces_nexp> returns a number > -1, a structural printout is delivered.

The number of spaces defines the tabulator distance. If <spaces_nexp> is -1 or not given, a compact printout is returned.

There may be some post-processing needed, especially if JSON objects beginning with a "{" and ending with a "}" are involved.

Note that multi-dimensional arrays are converted into **one-dimensional** arrays!

Example:

```
Bundle.GJ bptr, jsonString$
Byte.open w, ftb, path$+fileName$
Byte.write.buffer ftb, jsonString$
Byte.close ftb
```

See also: **JsonToXml$()**, **Is_Json()**

10.8 Bundle.GL

Syntax: Bundle.GL <pointer_nexp>, <key_sexp>, <list_ptr_nexp>

Short for Bundle.Get.List.

Places the list specified by the key string expression into the pointer specified list. The type (string or numeric) of the destination list must match the type stored with the key. If the bundle does not exist or does not contain the requested key, the command generates a run-time error.

Example:

```
Bundle.GL bptr,"names", listPtr
```

10.9 Bundle.GS

Syntax: Bundle.GS <pointer_nexp>, <key_sexp>, <stack_ptr_nexp>

Short for Bundle.Get.Stack.

Places the stack specified by the key string expression into the pointer specified stack. The type (string or numeric) of the destination stack must match the type stored with the key. If the bundle does not exist or does not contain the requested key, the command generates a run-time error.

Example:

```
Bundle.GS bptr,"toDos", stackPtr
```

10.10 Bundle.GV

Syntax: Bundle.GV <pointer_nexp>, <key_sexp>, <boolean_nval|Array[]>

Short for Bundle.Get.Verum. Verum is Latin for Truth.

Places the **boolean** specified by the key string expression into with <boolean_nval> specified numeric variable which can be also an array. If the **boolean** bundle content is **true** it returns 1. Otherwise (**false)** 0 is returned. If the bundle does not exist or does not contain the requested key, the command generates a run-time error.

Example:

```
Bundle.GV bptr,"boolean", myBoolean
```

10.11 Bundle.in

Syntax: Bundle.in <recAction_sexp>, <retData_svar>, <retBundleIndex_nvar>

Receives an Intent from a calling app or launcher. The parameter recAction returns the calling action. With retData you get the Data URI string from the Intent. The retBundleIndex returns a bundle. If no data is broadcasting retData returns "" and retBundleIndex returns an empty Bundle. Use DECODE$ with the type "URL" and the Qualifier "charset" like DECODE$ ("URL", "charset", retData$) if necessary.

Parameters in the retData String have to be in URI style, only.

The returned Bundle with the retBundleIndex pointer contains the received Extras.

10.12 Bundle.keys

Syntax: Bundle.keys <bundle_ptr_nexp>, <list_ptr_nexp>

Returns a list of the keys currently in the specified bundle.

The bundle pointer parameter <bundle_ptr_nexp> specifies the bundle from which to get the keys. If the bundle does not exist, a new one may be created.

The list pointer parameter <list_ptr_next> specifies the list into which to write the keys. The previous contents of the list are discarded. If the parameter does not specify a valid string list to reuse, and the parameter is a string variable, a new list is created and a pointer to the list is written to the variable.

The key names in the returned list may be extracted using the various list commands.

Example:

```
Bundle.keys bptr, list
List.size list, size
For i = 1 To size
 List.get list, i, key$
 Bundle.type bptr, key$, type$
 If type$ = "S" Then
  Bundle.get bptr, key$, value$
  Print key$, value$
 Else
  Bundle.get bptr, key$, value
  Print key$, value
 EndIf
Next i
```

10.13 Bundle.kill.last

Removes the last Bundle from the internal Bundles list. Since bundles are global, if you create a bundle within a function you can kill it before leaving the function.

10.14 Bundle.load

Syntax: Bundle.load <pointer_nexp>, <fileName_sexp>

Loads a bundle from a file. Only recommended for temporary use, because the internal coding depends on the operating system version.

Bitmaps in bundles are not supported. Maybe there is a size limitation. Not tested yet.

Example:

```
Bundle.load bptr, "saveState.bun"
```

10.15 Bundle.out

Syntax: Bundle.out <sendAction_sexp>, <sendData_sexp>, <sendExtraBundle_pointer_nexp>

Sends a result as an Intent return to a calling app or launcher, when the program is finished. The App or BASIC! has to be called **inclusive component name** (normally package name +".Basic"). If you launch the program in the Editor, you normally get **no result**.

The parameter <sendAction_sexp> sends an action back (extremely seldom). With <sendData_sexp>, you send a Data Extra string back (seldom). Android accepts only URIs! The <sendExtraBundle_pointer_nexp> sends a bundle back. This bundle contains the sent Extras.

When an app based on OliBasic is called by intent from another app and the BASIC program finishes, bundle.out returns a bundle to the caller.

In this case we have two bundle levels:

First level:

Supporting **only** the Intent Extras Types **Double, String and Bundle.**

Other types are automatically removed.

Second level:

If you use on the first level a bundle that contains also one or more bundles, you can use these as full featured BASIC! bundles.

See **App.SAR** for an example showing the use of **Bundle.out**.

10.16 Bundle.PB

Syntax: Bundle.PB <pointer_nexp>, <key_sexp>, <bundle_pointer_nexp>

Short for Bundle.Put.Bundle. Puts a bundle into a bundle.

The bundle <bundle_pointer_nexp> will be placed into with <pointer_nexp> specified bundle using the specified key. If the bundle specified <pointer_nexp> does not exist, a new one may be created.

The type of the value is "Bundle".

Example:

```
BUNDLE.PB bptr, "globals", bundlePtr
```

10.17 Bundle.PJ

Syntax: Bundle.PJ <pointer_nexp>, <json_sexp>

Short for Bundle.Put.JSON. Puts the content of a JSON string into a bundle.

The first level of the JSON content is parsed to bundle-supported types.

Arrays of JSON objects will be stored as Arrays of strings and have to be parsed outside the bundle, which should be also parsed as bundles.

Example:

```
jsonString$ = XmlToJson$("<name>Nicolas</name>") % Native XML string

! Workaround for a known bundle put issue: fill empty Arrays with a "" member
jsonString$ = Replace$(jsonString$, "[]", "[" + "\"" + "\"" +"]")

! Now fill empty Strings with  n.a.m. = "not a member"
jsonString$ = Replace$(jsonString$, "\"" + "\"", "\"" + "n.a.m."+ "\"")

Bundle.PJ bptr, jsonString$
Bundle.PJ bptr, "{\"age\":38, \"city\":\"Paris\"}" % Native JSON string
Bundle.PV bptr, "male", 1 % Boolean is true

preJSON$ = " 'street' : 'Main Street' , 'no' : 12 , 'FR' : false “

% Chr$(34) equals "\""
Bundle.PJ bptr, "{" + Replace$(preJSON$, "'", Chr$(34)) + "}"
Bundle.GJ bptr, newJson$, -1 % Returns a compact JSON string
Print newJson$
!! Prints the result
"{"street":"Main
Street","US":"true","no":12,"age":38,"city":"Paris","male":true,"name":"Nicolas"
}"
!!
```

See also XmlToJsonl$(), Is_Json(), Is_Xml()

10.18 Bundle.PL

Syntax: Bundle.PL <pointer_nexp>, <key_sexp>, <list_ptr_nexp>

Short for Bundle.Put.List. Puts a list into a bundle.

The list will be placed into the specified bundle using the specified key. If the bundle does not exist, a new one may be created.

The type of the value is "List".

Example:

```
Bundle.PL bptr, "names", llistPtr
```

10.19 Bundle.PS

Syntax: Bundle.PS <pointer_nexp>, <key_sexp>, <stack_ptr_nexp>

Short for Bundle.Put.Stack. Puts a stack into a bundle.

The stack will be placed into the specified bundle using the specified key. I f the bundle does not exist, a new one may be created.

The type of the value is a stack pointer.

Example:

```
Bundle.PL bptr, "toDos", stackPtr
```

10.20 Bundle.PV

Syntax: Bundle.PV <pointer_nexp>, <key_sexp>, <boolean_nexp|Array[]>

Short for Bundle.Put.Verum (Verum is Latin for Truth.) Puts a **boolean** value or array into a bundle.

The **boolean** will be placed into the specified bundle using the specified key. If the bundle does not exist, a new one may be created.

The type of the **boolean** value is a numeric expression or array. If <boolean_nexp> or an entry of an array is > 0, the bundle entry will be saved as **true**. Otherwise **false** will be saved.

Example:

```
Bundle.PV bptr, "globals", 1
```

10.21 Bundle.put

Syntax: Bundle.put <pointer_nexp>, <key_sexp>, <value_nexp> | Array[] | <value_sexp> | Array$[] {, ...}

The value expression will be placed into the specified bundle using the specified key. Arrays can be used for the value expressions. If the bundle does not exist, a new one may be created.

The type of the value will be determined by the type of the value expression.

Example:

```
Bundle.put bptr, "first_name", "Frank"
Bundle.put bptr,"age", 44
```

10.22 Bundle.remove

Syntax: Bundle.remove <pointer_nexp>, <key_sexp>

Removes the key named by the string expression <key_sexp>, along with the associated value, from the bundle pointed to by the numeric expression <pointer_nexp>. If the bundle does not contain the key, nothing happens. If the bundle does not exist, a new one may be created.

10.23 Bundle.save

Syntax: Bundle.save <pointer_nexp>, <fileName_sexp>

Saves a bundle into a file. Only recommended for temporary use, because the internal coding of the result depends on the operating system version. It seems that newer Android versions are able to read older bundle files. This is important if the user want to upgrade the operating system. But this behavior is without any guarantee.

Bitmaps in bundles are not supported. Maybe there is a size limitation. Not tested yet.

Example:

```
Bundle.save bptr, "saveState.bun"
```

See also Bundle.PJ, Byte.write.buffer, Grabfile, Bundle.GJ

10.24 Bundle.type

Syntax: Bundle.type <pointer_nexp>, <key_sexp>, <type_svar>

Returns the value type (string or numeric) of the specified key in the specified string variable. The <type_svar> will contain an uppercase "N" if the type is numeric, or an uppercase "S" if the type is a string. If the bundle does not exist or does not contain the requested key, the command generates a run-time error.

For arrays, dimensions are appended as in S[6] or N[2,2,5,..]. Other types result in "List", "Stack or Object" and "Bundle". If you get "Stack or Object" and you are not sure, dump the Bundle with BUNDLE.DUMP.BUNDLE first. If in doubt, use the type Object.

Example:

```
Bundle.type bptr, "age", type$
Print type$                    % will print N
```

11 Clipboard Commands

Unless your app is the default input method editor (IME) or is the app that currently has focus, your app cannot access clipboard data on Android 10 or higher.

11.1 Clipboard.get

Syntax: Clipboard.get <svar>{, <type_sexp}

Copies the current contents of the clipboard into <svar>.

The optional <type_sexp> specifies the type of content. Available are: "_Text" (default), "_Html", "_ToHtml" and "_Styled". In case of "_Styled", any text that would be returned as HTML formatting will be returned as text with Android style spans. To convert the main content into HTML text use "_ToHtml".

11.2 Clipboard.info

Syntax: Clipboard.info <bundle_nvar>

Places a description of the current contents of the clipboard in the bundle <bundle_nexp>.

Table of description arguments		
Key	**Value**	**Description**
_Label	String	The label of the clipboard content. Often also null is returned.
_Timestamp	Numeric	Timestamp in milliseconds since 1970-01-01T00:00:00. Android 8.0+ is needed.
_MimeType_1	String	Mime type 1, if available
_MimeType_2	String	Mime type 2, if available

11.3 Clipboard.put

Syntax: Clipboard.put <sexp>{{{, <label_sexp>}, <type_sexp>}, <html_sexp>}

Places <sexp> into the clipboard.

An optional label can be set with the <label_sexp>. A description of the predefined content type, such as "text" or "html", is recommended, but you can also indicate the origin of the use. The optional <type_sexp> defines the content type like "_Text" (default), "_Html" and "_Uri".

Example:

```
Clipboard.put "Text without html", "two times text", "_Html", "Text with html"
Clipboard.info bPointer
Debug.On
Clipboard.dump.bundle bPointer
Clipboard.get c$, "_Text"
Print c$
Clipboard.get c$, "_Html"
Print c$
```

12 Console Input and Dialogs

With the **Select** command you present information in a list format that looks very much like the Output Console. If you prefer, you can use **Dialog.select** to present the same information in a new dialog window. Either way, when your program runs, you select an item from the list by tapping a line.

The other commands in this group all pop up new windows.

Input lets you type a number or a line of text as input to your program. **Dialog.message** presents a message with a set of buttons to let you tell your program what to do next.

Popup is different. It presents information in a small, temporary display. It is not interactive and requires no management in your program. You pop it up and forget it.

The Text.input command operates on larger blocks of text, and TGet simulates terminal I/O.

12.1 Input

Syntax: Input {<prompt_sexp>}, <result_var>{, {<default_exp>}{, <canceled_nvar>}{, <layout_bundle_nexp>}{, <sel_nval>}

Generates a dialog box with an input area and an **OK** button. When the user taps the button, the value in the input area is written to the variable <result_var>.

The <prompt_sexp> will become the dialog box title. If the prompt expression is empty ("") or omitted, the dialog box will be drawn without a title area.

If the return variable <result_var> is numeric, the input must be numeric, so the only key taps that will be accepted are 0-9, "+", "-" and ".". If <result_var> is a string variable, the input may be any string.

If a <default_exp> is given then its value will be placed into the input area of the dialog box. The default expression type must match the <result_var> type.

The variable <canceled_nvar> controls what happens if the user cancels the dialog, either by tapping the BACK key or by touching anywhere outside of the dialog box.

If you provide a <canceled_nvar>, its value is set to **false** (0) if the user taps the **OK** button, and **true** (1) if the users cancels the dialog.

If you do not provide a <canceled_nvar>, a canceled dialog is reported as an error. Unless there is an "OnError:" the user will see the messages:

```
Input dialog cancelled
Execution halted
```

If there is an "OnError:" label, execution will resume at the statement following the label.

The <result_var> parameter is required. All others are optional. These are all valid:

```
Input "prompt", result$, "default", isCanceled
Input , result$, "default"
Input "prompt", result$, , isCanceled
Input "prompt", result$
Input , result$
```

Note the use of commas as parameter placeholders.

The optional <sel_nval> returns the clicked button, otherwise it returns 0.

Note also that in some Android versions, in Graphic Mode the width of the box shrinks according to the text size. If needed, use some spaces at the end of the prompt.

The optional layout bundle <layout_bundle_nexp> controls the Dialog.message output layout:

Table of layout control options		
Key	**Value**	**Description**
_Style	_Default	Default theme
	_Dark	Dark theme and Autosize mode
	_Bright	Bright theme and Autosize mode
_Icon	String	File path of a header icon
_PositionH	_Left	
	_Center	Default
	_Right	
_PositionV	_Top	
	_Center	Default
	_Bottom	
_Message	String	A message with possible HTML text formatting.
_MessageSize	numeric	HTML formatting takes precedence.
_MessageColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Sets the color of the message. HTML formatting takes precedence.
_Button1	String	HTML formatted text. Overwrites **OK**!
_Button1Size	numeric	
_Button1Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Sets the color of the button text. HTML formatting takes precedence.
_Button2	String	HTML formatted text.
_Button2Size	numeric	

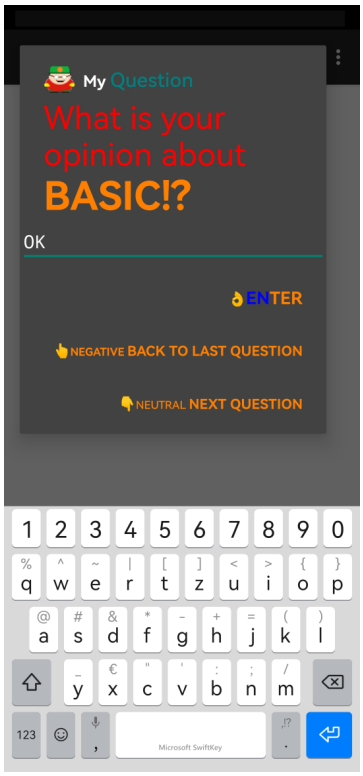
Table of layout control options		
Key	**Value**	**Description**
_Button2Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Sets the color of the button text. HTML formatting takes precedence.
_Button3	String	HTML formatted text.
_Button3Size	numeric	
_Button3Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Sets the color of the button text. HTML formatting takes precedence.
_Cancelable	numeric	If > 0 the dialog is cancel-able. Default is 1.
_MultiLine	numeric	If = 0 only a single line is supported. Default is 1.
_KbShow	numeric	If it is 1 the keyboard is shown automatically after calling. Otherwise the keyboard pops up if the screen is touched. Using a hardware keyboard hides the soft keyboard on writing. Default is 1.
_KbSuggestions	0 or 1 (numeric)	Switches the keyboard suggestions to off (0) and on (1). Keyboard suggestions make it often hard to write independent from a normal language. Default is 0.

Example

```
Bundle.put ptr, "_Style", "_Dark"
Bundle.put ptr, "_Icon", "cartman.png"
Bundle.put ptr, "_PositionH", "_Left"
Bundle.put ptr, "_PositionV", "_Top"
Q1$ = "What is your opinion about <big><b><Font color = '#FF8000'>BASIC!?"
Bundle.put ptr, "_Message", "<small>" + Q1$
Bundle.put ptr, "_MessageSize", 40
Bundle.put ptr, "_MessageColor", "_Red"
Bundle.put ptr, "_Button1", "👌<b><big>En<Font color = '#FF8000'>ter"
Bundle.put ptr, "_Button1Color", "_Blue"
Bundle.put ptr, "_Button2", ~
 "👇<small><Font color = '#FF8000'>neutral <b><big>Next Question"
Bundle.put ptr, "_Button3", ~
 "👆<b><small><Font color = '#FF8000'>negative <big>Back to   Last Question"
Bundle.put ptr, "_Cancelable", 1
Bundle.put ptr, "_MultiLine", 0
Bundle.put ptr, "_KbShow", 1
Bundle.put ptr, "_KbSuggestions", 0

Input "<small><b>My</b> <big><Font color = '#008080'>Question", ~
 result$, "OK", canceled, ptr, sel
```

```
Print result$, canceled, sel
```

The result is shown in the following screenshot:



12.2 Dialog.cust.call

Syntax: Dialog.cust.call <retBut_nval>, <retBnd_nval>, <message_sexp>, <title_sexp>{{{{, <button1_sexp>}, <button2_sexp>}, <button3_sexp>}, <bndPtr_nexp>}

Finishes the dialog description process and starts the dialog. The keystrokes are returned in <retBut_nval>. If <retBut_nval> is assigned a negative number before the command is called, a timer is set in milliseconds, and it counts down before it ends the dialog.

The results will be returned by the bundle <retBnd_nval>. See **Dialog.cust.open** for details.

A message can be set using <message_sexp>. Be careful when using it. Test your dialog in landscape format as well.

The title can be set by <title_sexp>.

The buttons will be optionally set by <button1_sexp>, <button2_sexp> and <button3_sexp>.

More layout specifications and predefined values can be set by the bundle <bndPtr_nexp>.

Table of options		
Key	**Value**	**Description**
_Style	_Default	Default theme
	_Dark	Dark theme and Autosize mode

Table of options		
Key	**Value**	**Description**
	_Bright	Bright theme and Autosize mode
_Icon	String	Filename of a header icon. The icon is shown only if <message_sexp> contains characters.
_PositionH	_Left	
	_Center	Default
	_Right	
_PositionV	_Top	
	_Center	Default
	_Bottom	
_TitleSize	numeric	
_TitleColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimiter} string) or #{hn}hnhnhn (hex string)	
_MessageSize	numeric	Textsize of the message. Default is 12.
_MessageColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimiter} string) or #{hn}hnhnhn (hex string)	Color of the message.
_Button1Size	numeric	
_Button1Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimiter} string) or #{hn}hnhnhn (hex string)	
_Button2Size	numeric	
_Button2Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimiter} string) or #{hn}hnhnhn (hex string)	
_Button3Size	numeric	

Table of options		
Key	**Value**	**Description**
_Button3Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimiter} string) or #{hn}hnhnhn (hex string)	
_Cancelable	numeric	If > 0 the dialog is cancel-able. Default is 1.
_KbShow	0 or 1 (numeric)	If it is 1 the keyboard is shown automatically after calling. Otherwise the keyboard pops up if the screen is touched. Using a hardware keyboard hides the soft keyboard on writing. Default is 1.
_SetFocus	numeric	Sets a focus for input. If it is 0 no focus will be set. Default is 1 if any edit field.

12.3 Dialog.cust.edit

Syntax: Dialog.cust.edit <ID_nval>, <text_sexp>, <hint_sexp>, <inputType_sexp>{{{{, <enabled_nvar>}, <textSize_nexp>}, <textStyle_sexp>}, <textColor_sexp>}

Sets a given EditText to show at position <ID_nval> in the specified layout.

A text have to be set with <text_sexp>.

The background hint can be specified with <hint_sexp>. It is strongly recommended when using the landscape orientation.

The following input type options are available:

"_None",

"_Text",

"_TextCapCharacters", "_TextCapWords", "_TextCapSentences", "_TextAutoCorrect",

"_TextAutoComplete", "_TextMultiLine", "_TextImeMultiLine", "_TextNoSuggestions", "_TextUri", "_TextEmailAddress", "_TextEmailSubject", "_TextShortMessage", "_TextLongMessage", "_TextPersonName", "_TextPostalAddress", "_TextPassword", "_TextVisiblePassword", "_TextWebEditText", "_TextFilter", "_TextPhonetic", "_TextWebEmailAddress", "_TextWebPassword",

"_Number",

"_NumberSignedDecimal", "_NumberSigned", "_NumberDecimal", "_NumberPassword",

"_Date", "_Datetime", "_Time"

If <enabled_nvar> is greater than 0, the input field can be edited. Default is 1.

The optional <textSize_nexp> sets the text size. Default is 12 dpi of the Android standard resolution of 160 dpi, which is converted to the current screen resolution automatically.

The optional <textStyle_nexp> sets the text style. Available are "_Normal", "_Bold", "_Italic" and "_Bold_Italic".

The optional <textColor_nexp> sets the text color. Default is the standard color of the chosen style in the Dialog.cust.call layout bundle.

The result is returned in <retBnd_nval> of the Dialog.cust.call command.

Example:

```
Dialog.cust.edit 3, "Joel Smith", "Name", "_Text", 1, 30, "_Normal","_Red"
```

12.4 Dialog.cust.image

Syntax: Dialog.cust.image <ID_nval>, <image_sexp>{{{, <imageHeight_nexp>}, <newSvgColor_sexp>}, <oldSvgColor_nexp>}

Sets a given ImageView to show at position <ID_nval> in the specified layout.

The file path of the image is set by <image_sexp>.

The optional <imageHeight_nexp> sets the image height. Default is 48 dpi of the Android standard resolution of 160 dpi, which is converted into the current screen resolution automatically. The size ratio of the image is retained. To get the default height of 48 dpi, use 0. The original size can be chosen if <imageHeight_nexp> is -1.

If the image is stored as a vector SVG file, you can change its color with <newSvgColor_sexp>. The color to replace is specified by <oldSvgColor_nexp>. Default is "#333333".

Use the OliBasic color notations like:

{Alpha,} Red,Green, Blue (comma delimited string)

or

_{Alpha,} ColorName ({comma delimited} string)

or

#{hn}hnhnhn (hex string)

Example:

```
Dialog.cust.image 2, "cartman.png"
```

12.5 Dialog.cust.open

Syntax: Dialog.cust.open <layout_sexp>{{{, <bgColor_sexp>}, <bgImage_sexp>}, <scroll_nexp>}

Opens the dialog description process. The layout is specified by <layout_sexp>.

If it is not a standard layout, like a picker, each layout can have up to 100 static IDs, which can be switched to be shown dynamically.

The optional background color is set by <bgColor_sexp>:

{Alpha,} Red, Green, Blue (comma delimited string)

or

_{Alpha,} ColorName ({comma delimited} string)

or

#{hn}hnhnhn (hex string)

An optional background image file can be set by <bgImage_sexp>.

A scroll bar can be enabled by <scroll_nexp> if it is > 0. Default is 1.

Table of layouts	
Layout	**Description**
_TimePicker	Shows a system time picker. Returns a bundle named _TimePicker. This bundle returns the keys _Hour and _ Minute.

`Dialog.cust.open "_TimePicker"`
`Dialog.cust.call retBut, retBundle, "", "Time Picker", "OK", "Cancel"`
`Bundle.GB retBundle, "_TimePicker", tpBundle`
`Bundle.get tpBundle, "_Hour", hour`
`Bundle.get tpBundle, "_Minute", minute`
`Print Right$("0" + Int$(hour), 2); ":"; Right$("0" + Int$(minute), 2)`	

_DatePicker	Shows a system date picker. Returns a bundle named _DatePicker. This bundle returns the keys _Year, _Month and _DayOfMonth.

`Dialog.cust.open "_DatePicker"`
`Dialog.cust.call retBut, retBundle, "", "Date Picker", "OK", "Skip"`
`Bundle.GB retBundle, "_DatePicker", dpBundle`
`Bundle.get dpBundle, "_Year", year % year$ is also possible`
`Bundle.get dpBundle, "_Month", month % month$ is also possible`
`Bundle.get dpBundle, "_DayOfMonth", dayOfMonth`
`Print Int$(year); "-"; Right$("0" + Int$(month), 2); "-"; Right$("0" + Int$ (dayOfMonth), 2)`	

_Calendar	Shows a system calendar. Returns a bundle named _Calendar. This bundle returns the key _Date. The date is the time since 1970-01-01 00:00 in milliseconds. The date can be predefined using the key _SetDate in the layout bundle in the following Dialog.cust.call command.

`Bundle.put bndPtr, "_SetDate", INT$(TIME(2022, 12, 25, 12, 0, 0))`
`Dialog.cust.open "_Calendar"`
`Dialog.cust.call retBut, retBundle, "", "Calendar", "OK", "Skip", "", bndPtr`
`Bundle.GB retBundle, "_Calendar", cBundle`
`Bundle.get cBundle, "_Year", year % year$ is also possible`
`Bundle.get cBundle, "_Month", month % month$ is also possible`
`Bundle.get cBundle, "_DayOfMonth", dayOfMonth`
`Print Int$(year); "-"; Right$("0" + Int$(month), 2); "-"; Right$("0" + Int$ (dayOfMonth), 2)`	

_AnalogClock	Shows a system analog clock.

`Dialog.cust.open "_AnalogClock"`
`retBut = -4000 % Turns off the clock in 4 seconds`
`Dialog.cust.call retBut, retBundle, "", "Analog Clock", "OK"`	

_Login	Shows a login form. 1 ImageView, 2 EditText [User] 3 ImageView, 4 EditText [Password]

Table of layouts	
Layout	**Description**

`Dialog.cust.open "_Login"`
`Dialog.cust.image 1, "cartman.png"`
`Dialog.cust.edit 2, "", "User", "_Text",1, 30, "", "_Blue"`
`Dialog.cust.image 3, "galaxy.png"`
`Dialog.cust.edit 4, "", "Password", "_TextPassword",1, 30, "", "_Red"`
`Dialog.cust.call retBut, retBundle, "", "Please login!", "Ok", "Cancel"`
`! If the returned result of an EditText is a number, a number variable can also be used.`
`Bundle.get retBundle, "_Edit 2", user$`
`Bundle.get retBundle, "_Edit 4", password$`
`Print user$, password$`	

_EditRecord	Shows an edit form. 1 TextView (max. 50) 2 EditText (max. 50) 3 T…
_ImageEditRecord	Shows an edit form. 1 TextView (max. 33) 2 ImageView (max. 33) 2 EditText (max. 33) 4 T…
_Dialog0	Dummy layout
_Dialog1	Customizable layout
_Dialog2	Customizable layout
_Dialog3	Customizable layout
_Dialog4	Customizable layout
_Dialog5	Customizable layout
_Dialog6	Customizable layout
_Dialog7	Customizable layout
_Dialog8	Customizable layout
_Dialog9	Customizable layout
_Dialog10	Customizable layout
_Dialog11	Customizable layout

12.6 Dialog.cust.text

Syntax: Dialog.cust.text <ID_nval>, <text_sexp>{{{, <textSize_nexp>}, <textStyle_sexp>}, <textColor_sexp>}

Sets a given TextView to show at position <ID_nval> in the specified layout.

The optional <textSize_nexp> sets the text size. Default is 12 dpi of the Android standard resolution of 160 dpi, which is converted into the current screen resolution automatically.

The optional <textStyle_nexp> sets the text style. Available are "_Normal", "_Bold", "_Italic" and "_Bold_Italic".

The optional <textColor_nexp> sets the text color. Default is the standard color of the chosen style in the Dialog.cust.call layout bundle.

Example:

```
Dialog.cust.text 1, "Name", 16, "_Bold", "_Red"
```

12.7 Dialog.message

Syntax: Dialog.message {<title_sexp>}, {<message_sexp>}, <sel_nvar> {{{{, <button1_sexp>}, <button2_sexp>}, <button3_sexp>}, <layout_bundle_nexp>}

Generates a dialog box with a title, a message, and up to three buttons. When the user taps a button, the number of the selected button is returned in <sel_nvar>. If the user taps the screen outside of the message dialog or presses the BACK key, then the returned value is 0.

The string <title_sexp> becomes the title of the dialog box. The string <message_sexp> is displayed in the body of the dialog, above the buttons. The strings <button1_sexp> (positive), <button2_sexp> (neutral), and <button3_sexp> (negative) provide the labels for the buttons.

You may have 0, 1, 2, or 3 buttons. On most devices, the buttons are numbered from right-to-left, because Android style guides recommend the positive action on the right and the negative action on the left. Some devices differ. On compliant devices, tapping the right-most button returns 1.

All of the parameters except the selection index variable <sel_nvar> are optional. If any parameter is omitted, the corresponding part of the message dialog is not displayed. Use commas to indicate omitted parameters.

Examples:

```
Dialog.message "Hey, you!", "Is this ok?", ok, "Sure thing!", "Don't care", ~
 "No way!"
Dialog.message "Continue?", , go, "YES", "NO"
Dialog.message , "Continue?", go, "YES", "NO"
Dialog.message , , b
```

The first command displays a full dialog with a title, a message, and three buttons.

The second command displays a box with a title and two buttons – note that the **YES** button will be on the right and the **NO** button on the left. The third displays the same information, but it looks a little different because the text is displayed as the message and not as the title. Note the commas.

The fourth command displays nothing at all. The screen dims and your program waits for a tap or the BACK key with no feedback to tell the user what to do.

If <sel_nvar> is negative at the command start, the message dialog will be finished in <sel_nvar> milliseconds. In this case <sel_nvar> returns 0.0.

Note that in some Android versions in Graphic Mode, the width of the box shrinks according to the text size. Some spaces at the end of the title or the message will help if needed.

The <title_sexp> and <message_sexp> also support HTML code. So a line break needs a "
" instead of "\n". See the SELECT layout description table under _TextHtml. Some characters have to be changed, such as "<" to "<" and ">" to ">".

The optional layout bundle <layout_bundle_nexp> controls the **Dialog.message** output layout:

Table of layout control options		
Key	**Value**	**Description**
_Style	_Default	Default theme
	_Dark	Dark theme and Autosize mode
	_Bright	Bright theme and Autosize mode
_Icon	String	File path of a header icon
_PositionH	_Left	

Table of layout control options		
Key	**Value**	**Description**
	_Center	Default
	_Right	
_PositionV	_Top	
	_Center	Default
	_Bottom	
_MessageSize	numeric	
_MessageColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_Button1Size	numeric	
_Button1Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_Button2Size	numeric	
_Button2Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_Button3Size	numeric	
_Button3Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_Cancelable	numeric	If > 0 the dialog is cancel-able. Default is 1.

12.8 Dialog.multi

Syntax: Dialog.multi <retBut_nval>, <retChk_Array[]>, <items_Array$[]>, <title_sexp>{{{{, <button1_sexp>}, <button2_sexp>}, <button3_sexp>}, <bndPtr_nexp>}

Starts a multi-selection dialog. The keystrokes are returned in <retBut_nval>. If <retBut_nval> is assigned a negative number before the command is called, a timer is set to count down in milliseconds

before it ends the dialog.

The results will be returned in the array <retChk_Array[]>. This array has to be predefined with given checks.

The items of the list are given by <items_Array$[]>.

The length of both arrays have to be equal.

The buttons will be optionally set by <button1_sexp> (positive), <button2_sexp> (neutral) and <button3_sexp> (negative).

More layout specifications can be set by the bundle <bndPtr_nexp>. See the tables of **Dialog.cust.call** for more information.

Example:

```
Array.load items$[], "Kurt", "Franz", "Maria", "Luca", "Marc", "Nicolas", ~
 "Tanya", "Bob"
Array.load retChk[], 0, 1, 0, 1, 0, 1, 0, 1
Dialog.multi retBut , retChk[], items$[], "Title", "OK", "Chancel"
Print retBut
Print "Checked: ", retChk[1], retChk[2], retChk[3], retChk[4], retChk[5], ~
 retChk[6];
Print retChk[7], retChk[8]
```

12.9 Dialog.select

Syntax: Dialog.select <sel_nvar>, <Array$[]>|<list_nexp> {{, <title_sexp>}, <layout_bundle_nexp>}

Generates a dialog box with a list of choices for the user. When the user taps a list item, the index of the selected line is returned in the <sel_nvar>. If the user taps the screen outside of the selection dialog or presses the BACK key, then the returned value is 0.

<Array$[]> is a string array that holds the list of items to be selected. The array is specified without an index but must have been previously dimensioned or loaded via **Array.load**.

As an alternative to an array, a string-type list may be specified in the <list_nexp>.

The <title_sexp> is an optional string expression that will be displayed at the top of the selection dialog. If the parameter is not present, or the expression evaluates to an empty string (""), the dialog box will be displayed with no title.

The <title_sexp> supports also HTML code. See the SELECT layout description table under _TextHtml.

The optional layout bundle <layout_bundle_nexp> controls the **Dialog.select** output layout:

Table of layout control options		
Key	**Value**	**Description**
_Style	_Default	Default theme
	_Dark	Dark theme and Autosize mode
	_Bright	Bright theme and Autosize mode
_Icon	String	File path of a header icon
_PositionH	_Left	
	_Center	Default
	_Right	

Table of layout control options		
Key	**Value**	**Description**
_PositionV	_Top	
	_Center	Default
	_Bottom	
_Button1	Name of Button 1	**Positive** button, Returns -1
_Button1Size	numeric	
_Button1Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_Button2	Name of Button 2	**Neutral** button, Returns -2
_Button2Size	numeric	
_Button2Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_Button3	Name of Button 3	**Negative** button, Returns -3
_Button3Size	numeric	
_Button3Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delim.} string) or #{hn}hnhnhn (hex string)	
_Cancelable	numeric	If > 0 the dialog is cancel-able. Default is 1.

12.10 Dialog.single

Syntax: Dialog.single <retBut_nval>, <retSel_nvar>, <items_Array$[]>, <title_sexp>{{{{, <button1_sexp>}, <button2_sexp>}, <button3_sexp>}, <bndPtr_nexp>}

Starts a single-selection dialog. The keystroke is returned in <retBut_nval>. If <retBut_nval> is assigned a negative number before the command is called, a timer is set and it counts down in milliseconds before ending the dialog.

The result will be returned in <retSel_nvar>. This variable can be predefined with a given value.

The items of the list are given by <items_Array$[]>.

The buttons will be optionally set by <button1_sexp> (positive), <button2_sexp> (neutral) and <button3_sexp> (negative).

More layout specifications can be set by the bundle <bndPtr_nexp>. See the tables for **Dialog.cust.call** for more information.

Example:

```
Array.load stdColors$[], "_Black", "_White", "_Gray", "_Red", "_Green", ~
 "_Blue", "_Cyan"
retSel = 4
Dialog.single retBut, retSel, stdColors$[], "Choose a Color", "OK"
Print stdColors$[retSel]
```

12.11 Select

Syntax: Select <sel_nvar>, <Array$[]>|<list_nexp>, <title_sexp>{{{, <message_sexp>}, <press_nvar> }, <layout_bundle_nexp>}

The Select command generates a new screen with a list of choices for the user. When the user taps a screen line, the index of the selected line is returned in the <sel_nvar>. If the user presses the BACK key, then the returned value is 0.

<Array$[]> is a string array that holds the list of items to be selected. The array is specified without an index but must have been previously dimensioned, loaded via **Array.load**, or created by another command.

As an alternative to an array, a string-type list may be specified in the <list_nexp>.

The <title_sexp> is a string expression that is placed into the title bar at the top of the selection screen. If the expression evaluates to an empty string ("") the title is blank.

The <message_sexp> is an optional string expression that is displayed in a short Popup message. If the message is an empty string (""), there is no Popup. If the parameter is absent, the <title_sexp> string is used instead, but if the <title_sexp> is also missing or empty, there is no Popup.

If the optional <press_nvar> is present, the type of user touch a short tap (0), a long press (1) or a double tap (2) is returned in <press_nvar>. The delay for detecting the double tap corresponds to the default Android system settings.

Use commas to indicate omitted optional parameters.

If <sel_nvar> is negative when the command is executed, the **Select** dialog will end in <sel_nvar> * milliseconds. In this case <sel_nvar> returns 0.0.

The optional layout bundle <layout_bundle_nexp> controls the **Select** output layout:

Table of layout control options		
Key	**Value**	**Description**
_TextSize	numeric	
_TextColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Note, _TextFont or _TextStyle is needed also!

Table of layout control options		
Key	**Value**	**Description**
_TextBackgroundColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Has to be "0,0,0,0" if you want a background color, wallpaper or bitmap Note, _TextFont or _TextStyle is needed also!
_TextFont	_Default	
	_Serif	
	_Sans_Serif	
	_Monospace	
_TextStyle	_Normal	
	_Bold	
	_Bold_Italic	
	_Italic	
_TextHtml	0 or 1 (numeric)	Returns displayable styled text from the provided HTML string. But not all tags are supported. Any <img> tags in the HTML will display an image. Absolute ("file://") and relative paths are allowed. The image size has to be scaled before, because h= and w= are ignored. See _HtmlBitmapScale. Uses parts of TagSoup library to handle real HTML, including all of the brokenness found in the wild. <a href="..."> <b> <big>? <blockquote> <del> <cite> <dfn> ? Use instead chr$(1564) [Arabic Letter] at line begin for align=' right' <font size="..." color="..." face="..."> <h1>, <h2>, <h3>, <h4>, <h5>, <h6> <i> <img src="..."> <p> <small> <strike>? < A.7 <sub> <sup> <tt>? <u>

Table of layout control options		
Key	**Value**	**Description**
		Replace Space with , & with &, < with <, > with >, " with " if necessary.
_HtmlTextSelectable	0 or 1 (numeric)	Does only work in conjunction with _TextHtml, but the item selection works only with a **long** click.
_HtmlBitmapScale	-1, 0, > 0 (numeric)	Does only work in conjunction with _TextHtml. Scales the included bitmaps in the following ways: -1 No scaling, 0 (default) Only scaling proportional to the screen resolution > 0 Proportional to the font size If it is 1 the bitmap height is the same as the font size.
_DividerColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_DividerFilename	bitmap file path	
_DividerHeight	numeric	
_BackgroundWallpaper	0 or 1 (numeric)	Min. Jelly Bean 4.1 (API 16)
_BackgroundColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_BackgroundFilename	bitmap file path	Min. Jelly Bean 4.1 (API 16)
_Orientation	(numeric) -1, 0, 1, 2 or 3	The value sets the orientation of screen as follows: -1 = Orientation depends upon the sensors. 0 = Orientation is forced to Landscape. 1 = Orientation is forced to Portrait. 2 = Orientation is forced to Reverse Landscape. 3 = Orientation

Table of layout control options		
Key	**Value**	**Description**
		is forced to Reverse Portrait.
_SetSelection	(numeric)	Sets a preselected item. The item will not be selected but it will still be positioned appropriately. If the specified selection position is less than 1, then the item at position 1 will be selected.
_StackFromBottom	0 or 1 (numeric)	If 1 pin the view's content to the bottom edge, 0 to pin the view's content to the top edge
_Subtitle	String	Set the Action bar's subtitle.
_TitleShow	0 or 1 (numeric)	If 1 (default) Show the Action bar if it is not currently showing. It will resize application content to fit the new space available. If 0 Hide the Action bar if it is currently showing. It will resize application content to fit the new space available.
_TitleIcon	Icon file path	Add a large icon to the notification content view. http://romannurik.github.io/AndroidAssetStudio/index.html
_TitleHomeEnabled	0 or 1 (numeric)	Set whether to include the application home accordance in the Action bar. Home is presented as an activity icon. Have to be 1 if you want to show the icon. Have to be 0 if you want to hide the icon. The default setting is API dependent.
_TitleBackground	Background file path	
_TitleHtml	0 or 1 (numeric)	Returns displayable styled text from the provided HTML string. But not all tags are supported. Uses parts of TagSoup library to handle real HTML, including all of the brokenness found in the wild. <b> <big> <font size="..." color="..." face="..."> <h1>, <h2>, <h3>, <h4>, <h5>, <h6> <i> <small> <strike>? < A.7 <sub> <sup> <tt>? <u> Replace

Table of layout control options		
Key	**Value**	**Description**
		Space with , & with &amp, < with &lt, > with &gt, " with &quot if necessary. Usable for Title and Subtitle. Keep in mind that the Actionbar height will not be expanded.
_ShowStatusbar	0, 1 or 2 (numeric)	If 1 (default) The Status bar will be displayed. If 2 The Status bar will be transparent displayed. Min. Lollipop 5.0 (API 21) If 0 The Status bar will be hidden to the background. Min. Nougat 7.0 (API 24) Will be switched to option 2 or 1 if the current API level is lower.
_StatusbarColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Min. Lollipop 5.0 (API 21)
_StatusbarLight	0 or 1 (numeric)	If 0 (default) The Status bar background is dark. In this case the **bar content** will be **light**. If 1 The Status bar background is light. In this case the **bar content** will be **dark**. Min. Lollipop 5.0 (API 21)
_ShowNavigationbar	0, 1 or 2 (numeric)	If 1 (default) The Navigation bar will be displayed. If 2 The Navigation bar will be transparent displayed. Min. Lollipop 5.0 (API 21) If 0 The Navigation bar will be hidden to the background. Min. Nougat 7.0 (API 24) Will be switched to option 2 or 1 if the current API level is lower.
_NavigationbarColor	{Alpha,} Red, Green, Blue (comma delimited string)	Min. Lollipop 5.0 (API 21)

Table of layout control options		
Key	**Value**	**Description**
	or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_NavigationbarLight	0 or 1 (numeric)	If 0 (default), the Navigation bar background is dark. In this case the **bar content** will be **light**. If 1, the Navigation bar background is light. In this case the **bar content** will be **dark**. Minimum Lollipop 5.0 (API 21)

Note that you should scale the text size and the divider height in conjunction with the detected screen size.

If you want to change something in the layout bundle, it is sufficient to make the changes only in the bundle. Note that every change affects the whole thing.

Example:

```
Array.load months$[], "January", "February", "March", "April", "May", "June", ~
 "July", "August", "September", "October", "November", "December"
bundle.put layout,"_TextSize", 30
bundle.put layout,"_TextColor", "0,0,255"
bundle.put layout,"_TextBackgroundColor", "250,250,250,250"
bundle.put layout,"_TextStyle", "_Bold"
bundle.put layout,"_SetSelection", 10
! bundle.put layout,"_StackFromBottom", 1
! bundle.put layout,"_DividerFilename", "cartman.png"
bundle.put layout,"_DividerColor", "255,0,255,0"
bundle.put layout,"_DividerHeight", 5
! bundle.put layout,"_BackgroundFilename", "cartman.png"
tLong = 10
Select month, months$[], msg$, tLong ,layout
Print months$[month]
```

12.12 Text.input

Syntax: Text.input <svar>{, { <text_sexp>} , <title_sexp> }, <suggestions_nexp>}, <layout_bundle_nexp>}

This command is similar to the **Input** command except that it is used to input and/or edit a large quantity of text. It opens a new window with scroll bars and full text editing capabilities. You may set the title of the new window with the optional <title_sexp> parameter.

If the optional <text_sexp> is present then that text is loaded into the text input window for editing. If <text_sexp> is not present then the **Text.input** text area will be empty. If <title_sexp> is needed but **Text.input** text area is to be initially empty, use two commas to indicate the <title_sexp> specifies the title and not the initial text. <title_sexp> accepts HTML text for formatting.

When done editing, tap the Finish item in the menu. The edited text is returned in <svar>.

If you tap the BACK key or the STOP item in the menu, then all text editing is discarded. <svar> returns the original <sexp> text.

The optional parameter <suggestions_nexp> sets an input type. If it 1, text suggestion is switched on. Default is 0, text suggestion switched off.

The following example grabs the Sample Program file, **f01_commands.bas**, to string s$. It then sends s$ to **Text.input** for editing. The result of the edit is returned in string r$. r$ is then printed to console.

```
GrabFile s$, "../source/ Sample_Programs/f01_commands.bas"
Text.input r$,s$
Print r$
```

The optional layout bundle <layout_bundle_nexp> controls the **Text.input** layout:

Table of layout control options		
Key	**Value**	**Description**
_TextHtml	0 or 1 (numeric)	Returns displayable styled text from the provided HTML string. But not all tags are supported. Any <img> tags in the HTML will display an image. Absolute ("file://") and relative paths are allowed. The image size has to be scaled before, because h= and w= are ignored. Uses parts of TagSoup library to handle real HTML, including all of the brokenness found in the wild. <a href="..."> <b> <big>? <blockquote> <del> <cite> <dfn> ? Use instead chr$(1564) [Arabic Letter] at line begin for align=' right' <font size="..." color="..." face="..."> <h1>, <h2>, <h3>, <h4>, <h5>, <h6> <i> <img src="..."> <p> <small> <strike>? < A.7 <sub> <sup> <tt>? <u> Replace Space with , & with &, < with <, > with >, " with " if necessary.

12.13 TGet

Syntax: TGet <result_svar> {, <listPointer_nexp>}, <prompt_sexp> {{, <title_sexp>} {, <layout_bundle_nexp>}

Simulates a terminal. The current contents of the Output Console is displayed in a new window. The last line displayed starts with the prompt string followed by the cursor. The user types in the input and taps Enter. The characters that the user typed in are returned in <result_svar>. The prompt and response are displayed on the Output Console.

You may set the title of the text input window with the optional <title_sexp> parameter.

The prompt and response are displayed on the Output Console automatically, if the **<listPointer_nexp>** parameter is not used. This parameter points to a list of strings which was previously created. If you want the response in the list, use **List.add** with prompt and response after **TGet**.

The command has also three menu entries:

- Stop stops the command and returns an empty string.
- Clear clears the current chat session.
- Clip puts the displayed content into the clipboard.

You may set the title of the text input window with the optional **<title_sexp>** parameter. The title uses HTML coded text. For more information, see the **Select** command layout description.

The optional layout bundle **<layout_bundle_nexp>** controls the **TGet** output layout:

Table of layout control options		
Key	**Value**	**Description**
_TextColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Note: _TextFont or _TextStyle is needed also!
_BackgroundColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_KbShow	0 or 1 (numeric)	If 1, the keyboard is shown automatically. Otherwise the keyboard pops up if the screen is touched. Using a hardware keyboard hides the soft keyboard on writing. Default is 1.

Table of layout control options		
Key	**Value**	**Description**
_KbSuggestions	0 or 1 (numeric)	Switches the keyboard suggestions to off (0) and on (1). Keyboard suggestions make it often hard to write independent from a normal language. Default is 0.

13 Console Keyboard Commands

At the lowest level, you can use **Inkey$** to read raw keystrokes. You control display of the virtual keyboard with **Kb.hide**, **Kb.show**, and **Kb.toggle**.

13.1 InKey$

Syntax: InKey$ <svar>{{, <rawKeyEvent_svar>}, <utf-8_svar>}

Reports key taps for the a-z, 0-9, Space and the D-Pad keys. The key value is returned in **<svar>**.

The D-Pad keys are reported as "up", "down", "left", "right" and "go". If any key other than those have been tapped, the string "key nn" will be returned. Where nn will be the Android key code for that key.

If no key has been tapped, the "@" character is returned in **<svar>**.

Rapid key taps are buffered in case they come faster than the BASIC! program can handle them.

Keep in mind that **soft** keyboards send a limited character set. Characters like "°♤♡◇♧《》¡¿äöü" are only supported by **USB** or **Bluetooth** keyboards or other input devices like game pads.

The optional **<rawKeyEvent_svar>** returns a raw key event description with action, keyCode, scanCode, metaState, flags, repeatCount, eventTime, downTime, deviceId and source values.

The optional **<utf-8_svar>** returns the UTF-8 character.

If you want correct results, use **OnKey…:** interrupt handling instead a **Do** – **Until** loop.

Do not use the **Pause** command if have a lot keystrokes in conjunction with key event handling.

Example

```
KeyDown.on % The opposite is KEYDOWN.OFF
Do
Until 0

OnKeyDown:  % A Key Is Down interrupt

 ! The second string, raw key event parameter
 InKey$ mKey$, mKeyEvent$, mUniKeyEvent$
 Print "Got "; mKey$, "  "; mKeyEvent$, mUniKeyEvent$
 KeyDown.resume       % Resumes execution at the point the program was

! the OnKeyDown: interrupt occurred.
OnKeyPress:           % Imo OnKeyUp points the fact better
 InKey$ mKey$, mKeyEvent$, mUniKeyEvent$
 Print "Got "; mKey$, "  "; mKeyEvent$, mUniKeyEvent$
 Key.resume
```

13.2 KeyDown.on

Syntax: KeyDown.on

Turns on Key Down Detection.

13.3 KeyDown.off

Syntax: KeyDown.off

Turns off Key Down Detection. Default condition.

13.4 Kb.hide

Syntax: Kb.hide

Hides the soft keyboard.

If the keyboard is showing, and you have an **OnKbChange:** interrupt label, BASIC! will jump to your interrupt label when the keyboard closes.

The soft keyboard is always hidden when your program starts running, regardless of whether it is showing in the Editor.

Note: The soft keyboard is automatically hidden when you change screens. For example, if the keyboard is showing over the Output Console, and you execute **Gr.open** to start Graphics Mode, the keyboard is hidden. The keyboard will not be showing when you exit Graphics Mode and return to the Console. Similarly, if you show the keyboard over your Graphics screen and you execute **Gr.close** to return to the Console, the keyboard is hidden.

If you have an **OnKbChange:** interrupt label, automatically hiding the keyboard does **not** trigger a jump to the interrupt label.

13.5 Kb.resume

Syntax: Kb.resume

This statement should be placed at the end of the interrupt handler at **OnKbChange:**.

13.6 Kb.show

Syntax: Kb.show

Shows the soft keyboard.

If the keyboard is not showing, and you have an **OnKbChange:** interrupt label, BASIC! will jump to your interrupt label when the keyboard opens.

When the soft keyboard is showing, its keys may be read using the **Inkey$** command. The command may not work in devices with hard or slide-out keyboards.

You cannot show the soft keyboard over the Output Console unless you first **Print** to the Console.

13.7 Kb.showing

Syntax: Kb.showing <lvar>

Reports the visibility status of the soft keyboard. If the keyboard is showing, the <lvar> is set to 1.0 (true), otherwise the <lvar> is set to 0.0 (false).

This command reports only the status of the keyboard shown by **Kb.show**. For example, the keyboard attached to the **Input** command dialog box cannot be controlled by **Kb.show** or **Kb.hide** and its status is not reported by **Kb.showing**.

13.8 Kb.toggle

Syntax: Kb.toggle

Toggles the showing or hiding of the soft keyboard. If the keyboard is being shown, it will be hidden. If it is hidden, it will be shown.

13.9 Key.resume

Syntax: Key.resume

This statement should be placed at the end of the interrupt handler at **OnKeyPress:**.

13.10 KeyDown.resume

Syntax: KeyDown.resume

This statement should be placed at the end of the interrupt handler at **OnKeyDown:**.

13.11 OnKbChange:

Syntax: OnKbChange:

Label for an interrupt handler which traps a keyboard change. When done, execute the **Kb.resume** statement to resume the interrupted program.

If you show a soft keyboard with **Kb.show**, or close that keyboard with **Kb.hide** or by tapping the BACK key, the change takes some time. The keyboard may open or close a few hundred milliseconds after it is requested. **Kb.show** and **Kb.hide** block until the change is complete, but your program does not know when you tap the BACK key.

If you have an **OnKbChange:** interrupt label in your program, then when the keyboard changes as just described, but your program is interrupted and execution continues at the interrupt label.

This interrupt occurs only for keyboards you show with **Kb.show**. Keyboards attached to other screens, such as **TGet** or the **Input** dialog box, do not cause **OnKbChange:** interrupts.

If you show a soft keyboard with **Kb.show**, and you tap the BACK key, the keyboard closes. The current screen (Console or Graphics screen) does not close. BASIC! does not trap the keypress with either **OnBackKey:** or **OnKeyPress:**. You can use the **OnKbChange:** interrupt label to be notified that the keyboard closed.

13.12 OnKeyDown:

Syntax: OnKeyDown:

Label for an interrupt handler which traps a tap_down on any key except the volume keys. They have to be switched on with **VolKeys.On**.

13.13 OnKeyPress:

Syntax: OnKeyPress:

Label for an interrupt handler which traps a tap on any key. When done, execute the **Key.resume** statement to resume the interrupted program.

13.14 Volume Keys

Certain keys and buttons have special meaning to your Android device. By default, BASIC! propagates these special keycodes to the Android system, even if your program detects them. So, for example, when you press the "Volume Up" button, your program can catch it (see **Inkey$**), but you still see the Volume Control window open on your screen, and your audio gets louder.

The **VolKeys.off** and **VolKeys.on** commands let you control this behavior for five keys. These keys may be on the Android device or on a headset plugged into the device.

Volume Keys		
Key Name (Android docs)	**Key Code (decimal)**	**Usual Action**
VOLUME_UP	24	Increase speaker volume
VOLUME_DOWN	25	Decrease speaker volume
VOLUME_MUTE	164	Mute the speaker (Android 3.0 or later)
MUTE	91	Mute the microphone
HEADSETHOOK	79	Hang up a call, stop media playback

13.14.1 VolKeys.off

Syntax: VolKeys.off

Disables the usual action of the keys listed in the **Volume Keys** table. Your program can still detect these keypresses, but BASIC! does not pass the events on to the Android system.

13.14.2 VolKeys.on

Syntax: VolKeys.on

Enables the usual action of the keys listed in the **Volume Keys** table. This is the default setting when your BASIC! program starts.

14 Console Output Commands

There are four types of output screens: the Output Console, the Graphics Screen, the HTML Screen, and the Select Screen.

This section deals with the Output Console. See the section on Graphics for information about the Graphics Screen. See the section on HTML for information abut the HTML screen.

Information is printed to screen using the Print command. BASIC! Run-time error messages are also displayed on this screen.

There is no random access to locations on this screen. Lines are printed one line after the other.

Although no line numbers are displayed, lines are numbered sequentially as they are printed, starting with 1. These line numbers refer to lines of text output, not to locations on the screen.

14.1 Output Console and Power Consumption

The Output Console is often updated internally. Depending on the device, this process requires more or less power, especially in connection with bitmaps.

Use the **Select** command if possible.

The **Pause** command is also possible, but events are ignored during this period.

Behind the screens:

BASIC! creates a new console list view in its main loop each time, if the console buffer is not empty.

Starting with OliBasic XXII, in its main loop a new console list view is created only if the console buffer is not empty or the hash of the console buffer is changed. In other words, the content of the buffer has changed.

If you use the **Select** command, only a new **Select** list view is created and the main loop is paused until the selection is done and returns.

The difference is hot, slightly above body temperature and cool with **an older** HTC device.

By the way, if your device's body gets hot, you should replace the battery immediately because at this point the battery can destroy more than itself.

Therefore, it is good practice to use the **Select** command instead of the console with all BASIC! versions, if possible.

14.2 Cls

Syntax: Cls

Clears the Output Console screen.

14.3 Console.default

Syntax: Console.default

Resets the console layout to the default settings.

14.4 Console.front

Syntax: Console.front

Brings the Output Console to the front where the user can see it.

If BASIC! is running in the background with no screen visible, this command brings it to the foreground. If you have a different application running in the foreground, it will be pushed to the background.

If BASIC! is running in the foreground, but the Graphics or HTML screen is in the foreground, this command brings the Console to the foreground. BASIC! remains in Graphics or HTML mode.

If you get trouble bringing the application from background to front (Android < 5) use a user defined function ReorderToFront() function (see examples under **App.SAR**).

14.5 Console.isShown

Syntax: Console.isShown <lvar>

Returns 1.0 if the console is shown, otherwise returns 0.0.

If the interpreter starts before the console is established, in case of an APK option or OliBasic version, the status can be checked.

Example:

```
cis = 0     % Not needed if not used before.
Do
 Pause 10
 Console.IsShown cis
Until cis
```

14.6 Console.layout

Syntax: Console.layout <layout_bundle_nexp>

The layout bundle <layout_bundle_nexp> controls the console output layout:

Key	Value	Notes
_TextSize	Numeric	
_TextColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn} hnhnhn (hex string)	Must be used in conjunction with_TextFont or _TextStyle.
_TextBackgroundColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn} hnhnhn (hex. string)	Has to be "0,0,0,0" if you want a background color, wallpaper or bitmap. Must be used in conjunction with_TextFont or _TextStyle.
_TextFont	_Default	

Key	Value	Notes
	_Serif _Sans_Serif _Monospace	
_TextStyle	_Normal _Bold _Bold_Italic _Italic	
_TextHtml	0 or 1 (numeric)	Returns displayable styled text from the provided HTML string. But not all tags are supported. Any <img> tags in the HTML will display an image. Absolute ("file://") and relative paths are allowed. The image size must be scaled before, because h= and w= are ignored. See _HtmlBitmapScale. Uses parts of TagSoup library to handle real HTML, including all of the brokenness found in the wild. <a href="..."> <b> <big>? <blockquote> <cite> <del> <dfn> ? Use instead chr$(1564) [Arabic Letter] at line begin for align=' right' <font size="..." color="..." face="..."> <h1>, <h2>, <h3>, <h4>, <h5>, <h6> <i> <img src="..."> <p> <small> <strike>? < A.7 <sub> <sup> <tt>? <u> Replace Space with , & with &, < with <, > with >, " with " if necessary.
_HtmlTextSelectable	0 or 1 (numeric)	Only works in conjunction with _TextHtml, but the item selection works only with a **long** click.

Key	Value	Notes
_HtmlBitmapScale	-1, 0, > 0 (numeric)	Only works in conjunction with _TextHtml. Scales the included bitmaps in the following ways: -1 no scaling, 0 (default) only scaling proportional to the screen resolution > 0 proportional to the font size If it is 1 the bitmap height is the same as the font size.
_DividerColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn} hnhnhn (hex. string)	
_DividerFilename	bitmap file path	
_DividerHeight	numeric	
_BackgroundWallpaper	0 or 1 (numeric)	Minimum Jelly Bean 4.1 (API 16)
_BackgroundColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn} hnhnhn (hex. string)	
_BackgroundFilename	bitmap file path	Minimum Jelly Bean 4.1 (API 16)
_SetSelection	numeric	Sets a pre selected item. The item will not be selected but it will still be positioned appropriately. If the specified selection position is less than 1, then the item at position 1 will be selected.
_StackFromBottom	0 or 1 (numeric)	1 pins the view's content to the bottom edge, 0 pins the view's content to the top edge.

For simplicity, you can use the same layout bundle for **Console.layout** and the **Select** command. But the **_Orientation** key will be ignored. In this case also use **Console.orientation**.

If you want to change something in the layout bundle, it is sufficient to make the changes only in the bundle. Note that every change affects the whole thing. Place a layout change as soon as possible in the BASIC! Code. Maybe you have to place a little (**Pause**) break.

See also: **Console.orientation**, **Select**.

14.7 Console.line.count

Syntax: Console.line.count <count_nvar>

Sets the return variable <count_nvar> to the number of lines written to the Console. This command waits for any pending Console writes to complete before reporting the count.

14.8 Console.line.text

Syntax: Console.line.text <line_nexp>, <text_svar>

The text of the specified line number of the Console is copied to the <text_svar>.

14.9 Console.line.touched

Syntax: Console.line.touched <line_nvar> {, <touch_nvar>}

After an **OnConsoleTouch** interrupt indicates the user has touched the console, this command returns information about the touch.

The number of the line that the user touched is returned in the <line_nvar>.

If the optional <touch_nvar> is present, the type of user touch–a short tap or a long press–is returned in the <touch_nvar>. Its value will be 0 (false) if the touch was a short tap. Its value will be 1 (true) if the touch was a long press.

Its value will be 2 if the touch was a double tap. The delay for detecting the double tap conforms to the default Android system settings.

If <touch_nvar> returns numbers greater than 9 it provides swipe directions as follows:

10 → right, 11 ↓ down, 12 ← left, 13 ↑ up. If you want to select the line also, keep in mind, that the line has to be high enough. You can set the line height by **Console.layout** and its key **_TextSize**.

It seems that a long press (1) may not be detectable on newer Android versions. In that case, use the **Select** command. It is generally the better choice to select a ListView.

14.10 Console.orientation

Syntax: Console.orientation <nexp>

The value of the <nexp> sets the orientation of screen as follows:

-1 = Orientation depends upon the sensors.
0 = Orientation is forced to Landscape.
1 = Orientation is forced to Portrait.
2 = Orientation is forced to Reverse Landscape.
3 = Orientation is forced to Reverse Portrait.

You can monitor changes in orientation by reading the screen width and height using the **Screen** command.

See also: **GR.open**, **GR.orientation**, **Select**.

14.11 Console.save

Syntax: Console.save <filename_sexp>

The current contents of the Console is saved to the text file specified by the filename string expression.

If the graphics or HTML mode is enabled, an error occurs and asks for closing this mode.

See also: **Is_Gr**, **Is_HTML**

14.12 Console.screenshot

Syntax: Console.screenshot <filename_sexp> {,<quality_nexp>}

Saves the current screen to a file. The default path is "<pref base drive>/rfo-basic/data/". The file will be saved as a JPEG file if the filename ends in ".jpg". The file will be saved as a PNG file if the filename ends in anything else (including ".png").

The optional <quality_nexp> is used to specify the quality of a saved JPEG file. The value may range from 0 (bad) to 100 (very good). The default value is 50. The quality parameter has no effect on PNG files which are always saved at the highest quality level.

Note: The size of the JPEG file depends on the quality. Lower quality values produce smaller files.

14.13 Console.title

Syntax: Console.title {{<title_sexp>}, <options_bundle_nexp>}

Changes the title of the console window. If the <title_sexp> parameter is omitted, the title is changed to the default title, "BASIC! Program Output".

Starting with OliBasic 3.00 the console title is empty by default.

The optional options bundle <options_bundle_nexp> controls the Action and Navigation bar layouts:

Key	Value	Description
_Subtitle	String	Set the action bar's subtitle.
_TitleShow	0 or 1 (numeric)	If 1 (default), it shows the Action bar if it is not currently showing. It will resize application content to fit the new space available. If 0, it hides the Action bar if it is currently showing. It will resize application content to fit the new space available.
_TitleIcon	Icon file path	Add a large icon to the notification content view. http://romannurik.github.io/AndroidAssetStudio/index.html
_TitleHomeEnabled	0 or 1 (numeric)	Set whether to include the application home icon in the action bar. Home is presented as an activity icon. Set to 1 if to show the icon. Set to 0 if to hide the icon. The default setting is API dependent.
_TitleBackground	Background file path	The background of the title bar can be created by a bitmap file. Only one colored pixel is needed.
_TitleHtml	0 or 1 (numeric)	Returns displayable styled text from the provided HTML string. Not all tags are supported. Uses parts of TagSoup library to handle real HTML, including all of the brokenness found in the wild.

Key	Value	Description
		<b> <big> <font size="..." color="..." face="..."> <h1>, <h2>, <h3>, <h4>, <h5>, <h6> <i> <small> <strike>? < A.7 <sub> <sup> <tt>? <u> Replace Space with , & with &amp, < with &lt, > with &gt, " with &quot if necessary. Usable for Title and Subtitle. Keep in mind that the Actionbar height will not be expanded.
_StatusbarColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn} hnhnhn (hex string)	Minimum Lollipop 5.0 (API 21)
_StatusbarLight	0 or 1 (numeric)	If 0 (default), the Status bar background is dark and the **bar content** will be **light**. If 1, the Status bar background is light ant the **bar content** will be **dark**. Minimum Lollipop 5.0 (API 21)
_ShowNavigationbar	0, 1 or 2 (numeric)	If 1 (default), the Navigation bar will be displayed. If 2, the Navigation bar will be transparent. Minimum Lollipop 5.0 (API 21) If 0, the Navigation bar will be hidden to the background. Minimum Nougat 7.0 (API 24) Will be switched to option 2 or 1 if the current API level is lower.
_NavigationbarColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn} hnhnhn (hex string)	Minimum Lollipop 5.0 (API 21)

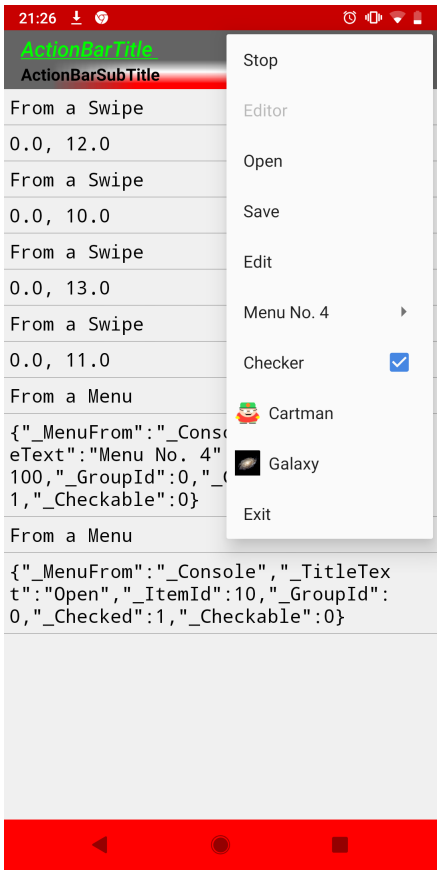
Key	Value	Description
_NavigationbarLight	0 or 1 (numeric)	If 0 (default), the Navigation bar background is dark and the **bar content** will be **light**. If 1, the Navigation bar background is light and the **bar content** will be **dark**. Minimum Lollipop 5.0 (API 21)
_Menu	Menu Bundle Pointer	Creates additional menu entries behind "Stop" and "Editor" if it is running in the development mode. A successful selection will be returned as a human readable JSON string.

Example

```
bundle.put c, "_Subtitle", " My Sub Title"
bundle.put c, "_TitleHomeEnabled", 1
bundle.put c, "_TitleBackground", "bgColor.png"
bundle.put c, "_TitleShow", 1
bundle.put c, "_TitleIcon", "cartman.png"

Console.title "My Console Title", c
Do
 Pause 50  % Saves power consumption!
Until 0
```

Example: How to Create a Menu

```
FN.DEF setMenuLayoutBundle()
 ! "Stop" default in development mode
 ! "Editor" default in development mode
 Bundle.put item10, "_Title", "Open"
 Bundle.PB menuLayout, "10", item10
 Bundle.put item30, "_Title", "Save"
 Bundle.PB menuLayout, "30", item30
 Bundle.put item40, "_Title", "Edit"
 Bundle.PB menuLayout, "40", item40
 Bundle.put item100, "_Title", "Menu No. 4"
 Bundle.put item100, "_SubMenuStart", 1
 Bundle.PB menuLayout, "100", item100
 Bundle.put item110, "_Title", "Sub Menu No. 1"
 Bundle.put item110, "_GroupId", 1
 Bundle.put item110, "_Checked", 1
 Bundle.PB menuLayout, "110", item110
 Bundle.put item120, "_Title", "Sub Menu No. 2"
 ! Next Line is important! Has to be placed behind
 ! the last SubMenu item.
 Bundle.put item120, "_GroupCheckable", 1
 ! Next Line is important! Has to be placed behind
 ! the last SubMenu item.
 Bundle.put item120, "_GroupExclusive", 1
 Bundle.put item120, "_GroupId", 1
 Bundle.PB menuLayout, "120", item120
 Bundle.put item200, "_Title", "Checker"
 ! Next Line is important! Has to be placed at the
 ! next entry behind the last SubMenu item.
 Bundle.put item200, "_AfterSubMenuEnd", 1
 Bundle.put item200, "_Checkable", 1
 Bundle.put item200, "_Checked", 1
 Bundle.PB menuLayout, "200", item200
 Bundle.put item210, "_Title", "Cartman"
 Bundle.put item210, "_Icon", "cartman.png"
 Bundle.PB menuLayout, "210", item210
```



```
 Bundle.put item220, "_Title", "Galaxy"
 Bundle.put item220, "_Icon", "galaxy.png"
 Bundle.PB menuLayout, "220", item220
 Bundle.put item230, "_Title", "Exit"
 Bundle.PB menuLayout, "230", item230
 Fn.rtn menuLayout
Fn.end
Bundle.PB layoutBundle , "_Menu", setMenuLayoutBundle()
```

Keep in mind, that only one sub menu level is possible on Android.

```
OnMenuItem:
 Print "From a Menu"
 MenuItem.get.dataLink data$
 Print data$
 ! TONE 600, 200
 If Is_in("Exit", data$) & Is_in("_Console", data$) Then End
 If Is_in("Exit", data$) & Is_in("_HTML", data$) Then Html.close : myCloser = 1
 If Is_in("Exit", data$) & Is_in("_Graphic", data$) Then Exit % GR.CLOSE :
myCloser = 1
 MenuItem.resume
```

14.14 ConsoleTouch.resume

Syntax: ConsoleTouch.resume

This statement should be placed at the end of the interrupt handler at **OnConsoleTouch:**

14.15 Popup

Syntax: Popup <message_sexp> {{, <x_nexp>}{, <y_nexp>}{, <duration_lexp>}}

Pops up a small message for a limited duration. The message is <message_sexp>.

All of the parameters except the message are optional. If omitted, their default values are 0. Use commas to indicate omitted parameters.

The simplest form of the **Popup** command, **Popup "Hello!"**, displays the message in the center of the screen for two seconds.

The default location for the Popup is the center of the screen. The optional <x_nexp> and <y_nexp> parameters give a displacement from the center. The values may be negative.

Select the duration of the Popup, either 2 seconds or 4 seconds, with the optional <duration_lexp> "long flag". If the flag is false (the expression evaluates to 0) the message is visible for 2 seconds. If the flag is true (non-zero) the message is visible for 4 seconds. If the flag is omitted the duration is short.

The popup is created outside the current app, in an Android message queue. These popup messages will be shown in any case step by step, even if the program has finished.

14.16 Print or ?

Syntax: Print {<exp> {,|;}} ... or ? {<exp> {,|;}} ...

Evaluates the expression(s) <exp> and prints the result(s) to the Output Console. You can use a question mark (**?**) in place of the command keyword **Print**.

If the comma (,) separator follows an expression, then a comma and a space will be printed after the value of the expression.

If the semicolon (**;**) separator is used, then nothing will separate the values of the expressions.

If the semicolon is at the end of the line, the output will not be printed until a **Print** command without a semicolon at the end is executed.

Print with no parameters prints a newline.

Examples:

```
Print "New", "Message"                 % Prints: New, Message
Print "New";" Message"                 % Prints: New Message
Print "New" + " Message"               % Prints: New Message

? 100-1; " Luftballons"                % Prints: 99.0 Luftballons
? Format$("##", 99); " Luftballons"    % Prints:   99 Luftballons

Print "A";"B";"C";                     % Prints: nothing
Print "D";"E";"F"                      % Prints: ABCDEF
```

Print with User-Defined Functions:

Print can operate on either strings or numbers. Sometimes it has to try both ways before it knows what to do. First it tries to evaluate an expression as a number. If that fails, it will try to evaluate the same expression as a string.

If this happens, and the expression includes a function, the function will be called twice. If the function has side-effects, such as printing to the console, writing to a file, or changing a global parameter, the side-effect action will also happen twice.

Be careful not to call a function, especially a user-defined function, as part of a **Print** command. Instead, assign the return value of the function to a variable, and then **Print** the variable. An assignment statement always knows what type of expression to evaluate, so it never evaluates twice.

```
! Do this:
y = MyFunction(x)
Print y

! NOT this:
Print MyFunction(x)
```

14.17 OnConsoleTouch:

Syntax: OnConsoleTouch:

Label for an interrupt handler which traps a tap on a line printed on the Output Console. When done, execute the **ConsoleTouch.resume** statement to resume the interrupted program.

You must touch a line written by a **Print** command, although it may be blank. Any touch in the empty area of the screen below the printed lines is ignored. After touching a Console line, you may use the **Console.line.touched** command to determine what line of text was touched.

This handler allows the user to interrupt an executing program in Console mode (not in Graphics mode). A common reason for such an interrupt would be to have the program request input via an **Input** statement.

To detect screen touches while in graphics mode, use **OnGrTouch:**.

15 Device Commands

These commands will give you information regarding your Android device and operating system.

Example:

```
Fn.def ViewSetting(setting$)
 List.create S, commandListPointer
 List.add commandListPointer ~
  "new Intent(" + Chr$(34) + setting$ + Chr$(34) + ");" ~
  "addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);" ~
  "addFlags(Intent.FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET);" ~
  "EOCL"
 List. create S, resultListPointer
 List.add resultListPointer, "EORL"
 Bundle.PL appVarPointer,"_CommandList",commandListPointer
 ! A Result List is needed if you want to wait until the dialog is finished.
 Bundle.PL appVarPointer,"_ResultList",resultListPointer

 App.SAR appVarPointer
 List.kill.last : List.kill.last : Bundle.kill.last % Clean up Lists and Bundles
Fn.end

! From
! https://developer.android.com/reference/android/provider/Settings#public-
constructors
! We get an action like ACTION_BLUETOOTH_SETTINGS
! we have to convert into android.settings.BLUETOOTH_SETTINGS

actionCode$ =  "ACTION_AIRPLANE_MODE_SETTINGS"
setting$ =  "android.settings." + REPLACE$(actionCode$, "ACTION_", "")
Bundle.clear myVarPointer
myVarPointer = ViewSetting(myVarPointer, setting$)

actionCode$ =  "ACTION_BLUETOOTH_SETTINGS"
setting$ =  "android.settings." + REPLACE$(actionCode$, "ACTION_", "")
Call ViewSetting (setting$)

actionCode$ =  "ACTION_PRIVACY_SETTINGS"
setting$ =  "android.settings." + REPLACE$(actionCode$, "ACTION_", "")
App.start setting$
Dialog.message "Dialog Finished", "", sel, "Ok" % Set a dialog box to wait.
```

15.1 Device

Syntax: Device <svar>

or

Syntax: Device <nexp>

Note: This command needs Phone permissions.

You can get information about your Android device with the **Device** command:

- The Device Brand, Device Model, Device Type, and OS
- The Language and Locale
- The PhoneType, PhoneNumber, and DeviceID
- The SN, MCC/MNC, and Network Provider stored on the SIM, if there is one.

The **Device** command has two forms that differ only in the type of the parameter, which determines the format of the returned data. Both forms return the same information, as shown in this table:

Key	Values	Meaning	Example (from emulator)
Brand	Any string	Brand name assigned by device manufacturer.	generic
Model	Any string	Model identifier assigned by device manufacturer.	sdk
Device	Any string	Device identifier assigned by device manufacturer.	generic
Product	Any string	Product identifier assigned by device manufacturer.	sdk
OS	OS Version	Android operating system version number.	4.1.2
Language	Language name	Default language of this device.	English
Locale	Locale code	Default locale code, typically language and country.	en_US
PhoneType	GSM, CDMA, SIP, None or Not Available	Type of phone radio in this device.	GSM
PhoneNumber	String of digits or Not Available	Phone number registered to this device, if any.	15555215554
DeviceID	String of digits or Not Available	The unique device ID, such as the IMEI, up to Android 8.1.	000000000000000
SIM SN	String of digits or Not Available	Serial number of the SIM card, if one is present and it is accessible.	89014103211118510720
SIM MCC/MNC	String of digits or Not Available	The "numeric name" of the provider of the SIM, if present and accessible.	310260
SIM Provider	Name string or Not Available	The name of the provider of the SIM, if present and accessible.	Android

The last six items access your device's telephony system and SIM card. If your device has no telephone, or BASIC! does not have permission to access them, the fields are set to neutral values: "None", "Not available", or a string of "0" characters.

In addition, there are convenience commands to retrieve only the Locale or the Language.

Information returned by **Device** is static. To get dynamic information, use the **Phone.Info** command.

The first form of this command returns information about your Android device in the string variable <svar>. Each item has this form:

```
key = value
```

The names `key` and `value` refer to the first two columns of the table in the **Device** overview. The items are placed in a single string, separated by newline characters. Formatted this way, if you **Print** the string it is displayed with one item on each line. You can separate the individual items with **Split**:

```
Device info$                          % all info in one string
Split info$[], info$, "\\n"           % each element is one item, "<k> = <v>"
Split lang_line$[], info$[6], " = "   % split Language item, two-element array
lang$ = lang_line$[2]                 % lang$ is language name, like "English"
```

The second form of this command returns information about your Android device in a Bundle pointed at by <nexp>. If you provide a variable that is not a valid Bundle pointer, the command creates a new Bundle and returns the Bundle pointer in your variable. Otherwise it writes into the Bundle your variable or expression points to.

The Bundle keys are shown in the first column of the table in the **Device** overview.

```
Device info                          % all info in a Bundle
Bundle.get info, "Language", lang$ % lang$ is the language name, like "English"
```

15.2 Device$

Syntax: svar$ = Device$(<key_sexp>)

Note: This is a function, not a command.

Returns information about the device, similar to the **Device.xx** subcommands. For this function you need **no** Phone permissions. You pass a key string in <key_sexp> to obtain the desired information, as shown in this table:

Key	Example	Description
_Language	English	Device language
_Locale	en_US	Location
_Os	4.0.3	Operating System version
_BattPercent	73	Current battery percentage
_BattStatus	Discharging	Battery status (Charging or Discharging)
_NightMode	NightNo	Can be NightNo, NightYes (Dark mode), NightAuto or ""

15.3 Device.auto.brightness

Syntax: Device.auto.brightness <bright_nexp>

Sets brightness mode to automatic. Needs WRITE_SETTINGS permission.

15.4 Device.get.brightness

Syntax: Device.get.brightness <brigth_nvar>

Returns the brightness in the range from 0 to 255. Needs WRITE_SETTINGS permission.

15.5 Device.language

Syntax: Device.language <svar>

Returns the default language of the device as a human-readable string value in string variable <svar>.

This convenience shortcut returns the same value as the "Language" key of the Bundle returned by the numeric form of the **Device** command.

15.6 Device.locale

Syntax: Device.locale <svar>

Returns the default locale code of the device in standard Locale format, typically including the language and country codes.

This convenience shortcut returns the same value as the "Locale" key of the Bundle returned by the numeric form of the **Device** command.

15.7 Device.os

Syntax: Device.os <api_nvar>{{{{{, <release_svar>} , <codename_svar>} , <incremental_svar>} , <security_svar>} , <base_os_svar>}

Returns the operating system version as a number. This command does not need Phone permissions.

<release_svar> returns the user-visible version string, such as "1.0" or "3.4b5".

<codename_svar> returns the current development code name, or the string "REL" if this is a release build.

<incremental_svar> returns the internal value used by the underlying source control to represent this build. For example, a perforce change list number or a git hash.

<security_svar> returns the user-visible security patch level. Needs API level 23.

<base_os_svar> returns the base OS build the product is based on. Needs API level 23.

15.8 Device.set.brightness

Syntax: Device.set.brightness <brigth_nexp>

Sets brightness mode to manual, and sets the brightness in the range of 0 to 255. Needs WRITE_SETTINGS permission.

15.9 Device.USB

Syntax: Device.USB <bundlePointer_nvar>

Returns the parameters of connected USB devices.

<bundlePointer_nvar> contains a bundle pointer. The bundle will contain a key for each USB device. The keys will be strings containing sequential integers. For example, if there are two USB devices, the first key will be "1" and the second key will be "2". The value associated with each key will be a collection of keys and values, resembling the structure of a bundle. So if there are two USB devices, the bundle might look like this:

```
1 : _DeviceId : xxx
    _VendorId : xxx
2 : _DeviceId : xxx
    _VendorId : xxx
```

You could say that **<bundlePointer_nvar>** points to a bundle of sub-bundles.

The individual sub-bundles can be extracted using the **Bundle.GB** command.

The sub-bundles may contain the following keys:

Key	Description
_DeviceId	Device ID
_VendorId	Vendor ID
_ProductId	Product ID
_DeviceClass	Device class
_InterfaceClass0	Interface class
_DeviceSubclassId	Device subclass ID
_DeviceName	Device name

Key	Description
_SerialNumber	Serial number
_ManufacturerName	Manufacturer name
_ProductName	Product name
_UsbInterface	USB interface
_Driver	Driver name
_AllAsString	All of the above, returned as a single string

Starting with Android 10, permission has to be granted for each USB device. If a dialog is canceled, or not confirmed within five seconds, the property enumeration is skipped.

Example:

```
Device.USB bp                          % Create a bundle from the USB devices.
Bundle.keys bp, lp                     % Make a keys list to get keys count.
List.size lp, n                        % Get number of keys.
List.clear lp                          % The list is no longer needed.
For i = 1 To n                         % Iterate through all keys.
 Bundle.GB bp, int$(i), pUSB           % Get a value and put it into a bundle.
 Bundle.get pUSB, "_AllAsString", aAS$ % Get a value from the new bundle.
 Print aAS$                            % Show it.
Next
```

Also see section 68 USB Commands for more on USB devices.

15.10 Phone.info

Syntax: Phone.info <nexp>|<nvar>

Returns information about the telephony radio in your Android device, if it has one. The information is placed in a Bundle. If you provide a variable that is not a valid Bundle pointer, the command creates a new Bundle and returns the Bundle pointer in your variable. Otherwise it writes into the Bundle your variable or expression points to.

The Bundle keys and possible values are in the table below. Each entry's type is either N (Numeric) or S (String).

Key	Type	Values	Meaning	Example
PhoneType	S	**GSM**, **CDMA**, **SIP**, or **None**	Type of phone radio in this device	GSM
NetworkType	S	**GPRS**, **EDGE**, **UMTS**, **CDMA**, **EVDOrev0**, **EVDOrevA**, **1xRTT**, **HSDPA**, **HSUPA**, **HSPA**, **iDen**, **EDVOrevB**, **LTE**, **EHRPD**, **HSPAP+**, or **Unknown**	Network type of the current data connection	LTE

The **PhoneType** is the same as that returned by the **Device** command. It is static.

If the **PhoneType** is **GSM** and the phone is registered to a network, the **Phone.info** command also returns the following items in the Bundle:

Key	Type	Values	Meaning	Example
CID	N	A positive number or -1 if CID is unknown	GSM Cell ID	342298497
LAC	N	A positive number or -1 if LAC is unknown	GSM Location Area Code	11090
MCC/MNC	S	String of 5 or 6 decimal digits	The "numeric name" of the registered network operator	310260
Operator	N	A name string	The name of the operator of the registered network	T-Mobile

The "numeric name" is made up of the Mobile Country Code (MCC) and Mobile Network Code (MNC).

If the **PhoneType** is **CDMA** and the phone is registered to a network, the **Phone.info** command also returns the following items in the Bundle:

Key	Type	Values	Meaning
BaseID	N	A positive number or -1 if BaseID is unknown	CDMA base station identification number
NetworkID	N	A positive number or -1 if NetworkID is unknown	CDMA network identification number
SystemID	N	A positive number or -1 if SystemID is unknown	CDMA system identification number

If your program has executed **Phone.rcv.init**, then **Phone.info** may be able to report the strength of the signal connecting your phone to the cell tower. If the signal strength is available, **Phone.info** will try to report it in one or two of the following bundle keys (the first three are mutually exclusive):

Key	Type	Values	Meaning
SignalLevel	N	A positive number 0 - 4	General measure of signal quality as shown in status bar. Higher is better.
GsmSignal	N	A positive number, 0 - 31 or 99 if unknown	"SignalLevel" unavailable, phone type is GSM, get GSM level instead.
CdmaDbm	N	A negative number, typically -90 (strong) to -105 (weak)	"SignalLevel" unavailable, phone type is CDMA, get raw power level in dBm instead.
SignalASU	N	0 - 31, 99 (most) 0 - 97, 99 (LTE)	"Arbitrary Strength Units", range depends on network type.

This information is not available on some Android devices, depending on the device manufacturer and your wireless carrier.

15.11 Screen

Syntax: Screen rotation, size[], realsize[], density

Returns information about your screen.
- Provide numeric variables to get the rotation and density of the screen.
- Provide numeric array variables to get the "application size" and "real size" of the screen. A size is returned as two values, width first and height second, in a two-element array. If the array exists, it is overwritten. Otherwise a new array is created.

All parameters are optional. Use commas to indicate omitted parameters.

rotation: The current orientation of your screen relative to the "natural" orientation of your device. The natural orientation may be portrait or landscape, as defined by the manufacturer. The return value is a number from 0 to 3. Multiply by 90 to get the rotation in degrees clockwise.

size[]: The size of the screen in pixels available for applications. This excludes system decorations. The width and height values reflect the current screen orientation.

realsize[]: The current real size of the screen in pixels, including system decorations. NOTE: this value is available only on devices running Android version 4.2 or later. On other devices, the values will be the same as in the **size[]** array parameter.

density: A standardized Android density value in dots per inch (dpi), usually 120, 160, or 240 dpi. This is not necessarily the real physical density of the screen. This value never changes.

15.12 Screen.rotation

Syntax: Screen.rotation <nvar>

Returns a number in the <nvar> parameter representing the current orientation of your screen relative to the "natural" orientation of your device. The natural orientation may be portrait or landscape, as defined by the manufacturer. The return value is a number from 0 to 3. Multiply by 90 to get the rotation in degrees clockwise.

See example under **Screen.size**.

15.13 Screen.size

Syntax: Screen.size size[], realsize[], density

Returns information about the size and density of your screen.

- You may provide numeric array variables to get the "application size" and "real size" of the screen. A size is returned as two values, width first and height second, in a two-element array. If the array exists, it is overwritten. Otherwise a new array is created.
- Provide a simple numeric variable to get the density of the screen.

All parameters are optional. Use commas to indicate omitted parameters.

size[]: The size of the screen in pixels available for applications. This excludes system decorations. The width and height values reflect the current screen orientation.

realsize[]: The current real size of the screen in pixels, including system decorations. NOTE: this value is available only on devices running Android version 4.2 or later. On other devices, the values will be the same as in the **size[]** array parameter.

density: A standardized Android density value in dots per inch (dpi), usually 120, 160, or 240 dpi. This is not necessarily the real physical density of the screen. This value never changes.

If your program is running in Graphics mode, "**Screen.size xy[], , dens**" returns the same values as "**Gr.screen x, y, dens**". Unlike **Gr.screen**, **Screen.size** also works in Console and HTML modes.

Example:

```
Screen.size size[], realsize[], density
Screen.rotation rotation
screenRatio = realsize[1] / realsize[2]

If screenRatio > 1 Then
 If Mod(rotation, 2) Then    % rotation = 1 | rotation = 3
  Bundle.put globals, "landscapeRot", 1
  Bundle.put globals, "portraitRot", 0
 Else                        % rotation = 0 | rotation = 2
  Bundle.put globals, "landscapeRot", 0
  Bundle.put globals, "portraitRot", 1
 EndIf
```

```
Else
 If Mod(rotation, 2) Then    % rotation = 1 | rotation = 3
  Bundle.put globals, "landscapeRot", 0
  Bundle.put globals, "portraitRot", 1
 Else                        % rotation = 0 | rotation = 2
  Bundle.put globals, "landscapeRot", 1
  Bundle.put globals, "portraitRot", 0
 EndIf
EndIf

Debug.on
Debug.dump.bundle globals
```

15.14 WiFi.info

Syntax: WiFi.info {{<SSID_svar>}{, <BSSID_svar>}{, <MAC_svar>}{, <IP_var>}{, <speed_nvar>}}

Gets information about the current Wi-Fi connection and places it in the return variables.

All of the parameters are optional; use commas to indicate omitted parameters. The table shows the available data:

Variable	Type	Returned Data	Format
SSID	String	SSID of current 802.11 network	"name" or hex digits (see below)
BSSID	String	BSSID of current access point	xx:xx:xx:xx:xx:xx (MAC address)
MAC	String	MAC address of your WiFi	xx:xx:xx:xx:xx:xx
IP	Numeric or String	IP address of your WiFi	Number or octets (see below)
speed	Numeric	Current link speed in Mbps	Number

Format notes:

- SSID: If the network is named, the name is returned, surrounded by double quotes. Otherwise the returned name is a string of hex digits.
- IP: If you provide a numeric variable, your Wi-Fi IP address is returned as a single number. If you provide a string variable, the number is converted to a standard four-octet string. For example, the string format 10.70.13.143 is the same IP address as the number -1887287798 (hex 8f82460a).

On newer Android systems, if you need <SSID_svar> and <BSSID_svar> your device needs access to Fine Location Permissions. On devices below Android 6 this code will also run, because Permission commands will be ignored.

```
Permission.get granted, "ACCESS_FINE_LOCATION"
If !granted Then Permission.request "ACCESS_FINE_LOCATION"
WiFi.info ssid$, bssid$, mac$, ip$, speed

Print ssid$, bssid$, mac$, ip$, speed
```

16 Debug Commands

The debug commands help you debug your program. The **Debug.on** command controls execution of all the debug commands. The debug commands are ignored unless the **Debug.on** command has been previously executed. This means that you can leave all your debug commands in your program and be assured that they will not execute unless you turn debugging on with **Debug.on**.

16.1 Debug.dump.array

Syntax: Debug.dump.array Array[]

Dumps the contents of the specified array. If the array is multidimensional the entire array will be dumped in a linear fashion.

16.2 Debug.dump.bundle

Syntax: Debug.dump.bundle <bundlePtr_nexp>

Dumps the Bundle pointed to by the Bundle Pointer numeric expression.

If a Bundle contains other types, such as single numeric values from type Double or String, this value will be converted and printed as a string without quotation marks.

16.3 Debug.dump.fn

Syntax: Debug.dump.fn {<level_nexp>}

Dumps the function name from the function stack.

<level_nexp> is the optional maximum level (default is 1):

1 - print only the last level (current function)
2 - print two levels (last + previous caller)
..etc...
0 - print all levels

16.4 Debug.dump.list

Syntax: Debug.dump.list <listPtr_nexp>

Dumps the List pointed to by the List Pointer numeric expression.

16.5 Debug.dump.scalars

Syntax: Debug.dump.scalars

Prints a list of all the Scalar variable names and values. Scalar variables are the variable names that are not Arrays or Functions. Among other things, this command will help expose misspelled variable names.

16.6 Debug.dump.stack

Syntax: Debug.dump.stack <stackPtr_nexp>

Dumps the Stack pointed to by the Stack Pointer numeric expression.

16.7 Debug.echo.off

Syntax: Debug.echo.off

Turns off the Echo mode. Same as **Echo.off**.

16.8 Debug.echo.on

Syntax: Debug.echo.on

Turns on Echo mode. Same as **Echo.off**. Each line of the running BASIC! program is printed before it is executed. This can be of great help in debugging. The last few lines executed are usually the cause of program problems. The Echo mode is turned off by either the **Debug.echo.off** or the **Debug.off** commands.

16.9 Debug.off

Syntax: Debug.off

Turns off debug mode. All debug commands (except **Debug.on**) will be ignored. When your program exits, the broken-loop checks are not performed.

16.10 Debug.on

Syntax: Debug.on

Turns on debug mode. All debug commands will be executed when in the debug mode.

Debug.on also enables a simple debugging aid built into BASIC!. If debug is on, and your program entered a loop but did not exit the loop cleanly, you will get a run-time error. See the looping commands (**For**, **While**, and **Do**) for details.

16.11 Debug.print

Syntax: Debug.print

This command is exactly the same as the **Print** command except that the print will occur only while in the debug mode.

16.12 Debug.show

Syntax: Debug.show

Pauses the execution of the program and displays a dialog box. The dialog box will contain the result of the last **Debug.show.<command>** used or by default **Debug.show.program**.

There are three buttons in the dialog:

Resume: Resumes execution.
Step: Executes the next line while continuing to display the dialog box.
View Swap: Opens a new dialog that allows you to choose a different Debug View.

The BACK key closes the debug dialog and stops your program.

16.13 Debug.show.array

Syntax: Debug.show.array Array[]

Pauses the execution of the program and displays a dialog box. The dialog box prints the contents of

the specified array, the line number of the program line just executed and the text of that line. If the array is multidimensional the entire array will be displayed in a linear fashion.

For a description of the dialog box controls, see the **Debug.show** command.

16.14 Debug.show.bundle

Syntax: Debug.show.bundle <bundlePtr_nexp>

Pauses the execution of the program and displays a dialog box. The dialog box prints the Bundle pointed to by the Bundle Pointer numeric expression, the line number of the program line just executed and the text of that line.

For a description of the dialog box controls, see the **Debug.show** command.

16.15 Debug.show.list

Syntax: Debug.show.list <listPtr_nexp>

Pauses the execution of the program and displays a dialog box. The dialog box prints the List pointed to by the List Pointer numeric expression, the line number of the program line just executed and the text of that line.

For a description of the dialog box controls, see the **Debug.show** command.

16.16 Debug.show.program

Syntax: Debug.show.program

Pauses the execution of the program and displays a dialog box. The dialog box shows the entire program, with line numbers, as well as a marker pointing to the last line that was executed.

Note: the debugger does not stop on a function call. The first line of the function is executed and the marker points to that line. When a **Fn.rtn** or **Fn.end** executes, the marker points to the function call.

For a description of the dialog box controls, see the **Debug.show** command.

16.17 Debug.show.scalars

Syntax: Debug.show.scalars

Pauses the execution of the program and displays a dialog box. The dialog box prints a list of all the Scalar variable names and values, the line number of the program line just executed and the text of that line. Scalar variables are the variable names that are not Arrays or Functions. Among other things, this command will help expose misspelled variable names.

For a description of the dialog box controls, see the **Debug.show** command.

16.18 Debug.show.stack

Syntax: Debug.show.stack <stackPtr_nexp>

Pauses the execution of the program and displays a dialog box. The dialog box prints the Stack pointed to by the Stack Pointer numeric expression, the line number of the program line just executed and the text of that line.

For a description of the dialog box controls, see the **Debug.show** command.

16.19 Debug.show.watch

Syntax: Debug.show.watch

Pauses the execution of the program and displays a dialog box. The dialog box lists the values of the variables being watched, the line number of the program line just executed and the text of that line.

For a description of the dialog box controls, see the **Debug.show** command.

16.20 Debug.watch

Syntax: Debug.watch var, ...

Gives a list of Scalar variables (not arrays) to be watched. The values of these variables will be shown when the Debug.show.watch command is executed. This command is accumulative, meaning that subsequent calls will add new variables into the watch list.

16.21 Echo.off

Syntax: Echo.off

Turns off the Echo mode. Same as **Debug.echo.off**.

16.22 Echo.on

Syntax: Echo.on

Turns on Echo mode. Same as **Debug.echo.off**.

16.23 GoTo.get.index

Syntax: GoTo.get.index <nvar>{, <lastChar_nvar>}

Returns the current execution command index in **<nvar>**. The last character of current execution command will be returned in **<lastChar_nvar>**.

16.24 GoTo.get.error.index

Syntax: GoTo.get.error.index <nvar>{, <lastChar_nvar>}

Returns the execution command index at the last error in **<nvar>**. The last character of current execution command will be returned by **<lastChar_nvar>**.

16.25 GoTo.set.index

Syntax: GoTo.set.index <nexp>

Jumps to the given execution command. If **<nexp>** is greater than the size of the command list, it jumps to the last command.

Example:

```
! : Print "ci", ci   counts as an extra command
GoTo.get.index ci, lastChar : Print "ci", ci
pring % A command with trouble
Print "OK"
End

OnError:
 GoTo.get.error.index eri
```

```
 Print eri
 GoTo.set.index eri + 1 % Go to the command behind (+1) the error
```

Example:

```
! How to get the program line from the above lastChar
! GetProgLineNum.bas
File.root path$, "_Source"
File.select progPath$, path$
Text.open r, ftb, progPath$
counter = 0
mChars = 0
Input "Insert character number", lastChar
Do
 counter ++
 Text.readln ftb , line$
 mChars = mChars + Len(line$) + 1    % + 1, because of LF
 Text.eof ftb, mEof
 If mChars >= lastChar Then mEof = 1
Until mEof
Text.close ftb
Print "Line: "; counter
Print line$
```

Example:

```
GoTo.get.index ci
Bundle.put gbp, "Try01", ci

Fn.def myFunc01 (gbp, data)
 Print "FN 01"
 result01 = data * 3/  % Returns an error
 Fn.rtn result01
 Bundle.put gbp, "NewResult", r
 Fn.rtn r
Fn.end

GoTo.get.index ci
Bundle.put gbp, "Catch01", ci - 3

GoTo.get.index ci
Bundle.put gbp, "Try02", ci

Fn.def myFunc02 (gbp, data)
 Print "FN 02"
 result02 = data * "2" % Returns an error
 Fn.rtn result02
 Bundle.get gbp, "NewResult", r
 Fn.rtn r
Fn.end

GoTo.get.index ci
Bundle.put gbp, "Catch02", ci - 3
Debug.on
Debug.dump.bundle gbp
res = myFunc02(gbp, 33)
Print res
res = myFunc01(gbp, 33)
Print res
Print "OK"
Do
 Pause 100
Until 0

End

OnError:
 GoTo.get.error.index eri
```

```
Print "Error at Execution Command", eri
Bundle.get gbp, "Try02", Try02
Bundle.get gbp, "Catch02", Catch02
If eri > Try02 & eri < Catch02 Then ~
 Bundle.put gbp, "NewResult", 44 : GoTo.set.index Catch02
Print GetError$()
End
```

Instead of a Bundle you can also use **Globals.fnimp**, **Fn.import**, **Tries[]**, **Catches[]** etc.

17 Document Commands

Android supports documents. An Android document can be a directory, a file, a database or something else. Each document contains content, which is provided by a content provider.

A path pointing to a content begins with "`content://`".

A content provider gives access to its own data to other applications also. It can as an example be a file system or a cloud storage like Google Drive.

OliBasic supports an Adoc Provider and a File Provider, but you can also provide data in a file saved in an **external** directory or supported cloud storage.

External means public file access. SD cards and USB sticks are **removable** data carriers.

Each valid document path gives you access to the document name and the default size, but not in all cases to the data. Handling documents is a little different from handling files. For better understanding, we will look at cloud storage first. Cloud storage is connected by a network.

If you use a mobile network, the link can be broken. To prevent data loss, you need exact status each time.

If you want a directory document, you have to use **Adoc.save** with a document type named "_Dir". A new file needs also **Adoc.save**. In this case, you can preset the document type and its name. Creating a new directory in its workflow is also possible. Now the document has to be filled by **Adoc.write**. This command uses strings. If you want to copy or save binary data, use "_ISO-8859-1" as the character set.

To try to read the stored content, we start **Adoc.open** to choose a document. If the network connection is not broken, we get a document path beginning with "`content://`". **Adoc.read** allows you to get the content as a string. If you want to copy a document, read and put the content into a string with the "`_ISO-8859-1`" character set, and write the content into a new document crated by the **Adoc.save** dialog with the same character set. Try to enumerate the existance of the created document.
If a document has to be renamed, use **Adoc.rename**.

A content path is different to a file path. Sometimes it looks like a readable file path, but in cases of cloud storage, last documents, downloads etc. it is coded or/and encrypted. Use **File.absolute** to convert the document path into an absolute file path. If it fails using a valid document path, it returns only the document name without any slash ("/"). Selecting a document in the Downloads document directory returns a file path, if the document name equals a first level file name. If it returns a valid absolute file path, you can operate on it with normal file commands. Otherwise, copy documents into files to handle them.

In case of Adoc (Documents) Providers, the proper permissions have to be granted. In some commands there is a <grant_perm_nexp> parameter which defaults to 1. So permissions will be granted by default. In case of File Providers, the <grant_perm_nexp> parameter should be set to 0. To control the access to your own providers, see also the command **PROVIDER**.

What is the outlook? Google should continue to take care of user's security. The user has to be involved to confirm actions with possible data risks. That is not what developers normally want. They like paths, which can be extend by easy readable directory and file names. The consequence is storing selected document paths permanently in a private file folder created by **File.root path$, "_Internal"**. However, there is no guarantee that the document path will be the same after a restart or an extended period of time between runs of the application.

Known issue: <startPath_sexp> does not work properly in all cases. If you have a removable device

like an SD-card "/storage/9016-4EF9", look that the device's content path ends with "%3A" like "content://com.android.externalstorage.documents/document/9016-4EF9%3A".

17.1 Adoc.delete

Syntax: Adoc.delete <success_nvar>, <documentPath_sexp>

Returns 1 if the document specified by the document path was deleted successfully. Otherwise, 0 is returned.

Minimum KitKat 4.4 (API 19) is needed.

17.2 Adoc.exists

Syntax: Adoc.exists <lvar>, <documentPath_sexp>{, <grant_perm_nexp>}

Reports if the <documentPath_sexp> directory or document exists. If the directory or document does not exist, <lvar> will be set to zero. If the file or directory does exist, <lvar> will be set to non-zero.

Required permissions are granted until revoked, or the app is uninstalled.

Minimum KitKat 4.4 (API 19) is needed.

Use <grant_perm_nexp> to control permission grants. If a Document Provider does not ask for permissions, use 0 (false). Default is 1 (true).

17.3 Adoc.get

Syntax: Adoc.get <documentPath_svar>|Array$[]{{, <startPath_sexp>}, <mimeType_sexp>}

Opens a system dialog for opening a document, and returns the selected path in <documentPath_svar>. If <documentPath_svar> returns an empty string (""), the operation was not successful. If an Array$[] is used, multiselection is possible. If the selection in this case failed, the first array item returns an empty string (""").

An existing starting path can be defined by <startPath_sexp>. An empty string tells the document browser to start at the default data path. If this attribute is not used, it starts at the last visited location.

Android stores each document into a database and notes the document type. To shrink the number of selectable documents, use <mimeType_sexp> to specify the desired types.

Unlike **Adoc.open**, only document providers specified for **reading** can be selected.

The possibility of choices is usually larger.

The returned URI will only be safe for read access.

17.4 Adoc.grab

Syntax: Adoc.grab <result_sexp>, <documentPath_sexp>{, <charSet_sexp>}

Puts the string content specified by <documentPath_sexp> into <result_sexp>.

For binary data, specify the <charSet_sexp> "_ISO-8859-1" instead of the default "_UTF-8".

Required permissions are granted until revoked, or the app is uninstalled.

You can choose between following character sets: "_US-ASCII", "_UTF-8", "_UTF-16", "_UTF-16BE", "_UTF-16LE" and "_ISO-8859-1".

Normally you will use this instead of **Adoc.read**, because you can read also only for reading specified content providers.

Keep in mind, that a broken network connection is not checked for.

17.5 Adoc.lastModified

Syntax: Adoc.lastModified <success_nvar>, <documentLastModified_nvar>, <documentPath_sexp>

Returns 1 if the last modification of the document specified by the document path was returned successfully. Otherwise, 0 is returned. <documentLastModified_nvar> returns the time when this document was last modified, measured in milliseconds since January 1st, 1970, midnight GMT.

Example:

```
File.root pp$ ,"_SourceSamples"
Adoc.path dp$,"file://"+ pp$ + "/" + "f01_commands.bas"
Adoc.lastModified success, lm, dp$
? Using$("", "%TF %tT" ,lm, lm) % Returns "2019-06-30 21:01:01"
```

17.6 Adoc.mimeType

Syntax: Adoc.mimetype <success_nvar>, <mimeType_svar>, <documentPath_sexp>

Returns 1 if the MIME type of the document specified by the document path was returned successfully. Otherwise, 0 is returned. <documentMimeType_nvar> returns the MIME type of this document.

17.7 Adoc.name

Syntax: Adoc.name <documentName_svar>, <documentPath_sexp>

Returns the human-friendly name of the document in the given document path. If it is not provided, then the name should default to the last segment of the documents's URI.

If an error occurs, an empty string "" is returned.

17.8 Adoc.open

Syntax: Adoc.open <documentPath_svar>|Array$[]{{, <startPath_sexp>}, <mimeType_sexp>}

Opens a system dialog for opening a document, and returns the selected path in <documentPath_svar>. If <documentPath_svar> returns an empty string (""), the operation was not successful. If an Array$[] is used, multiselection is possible. If the selection in this case failed, the first array item returns an empty string ("").

An existing starting path can be defined by <startPath_sexp>. An empty string tells the document browser to start at the default data path. If this attribute is not used, it starts at the last visited location.

Android stores each document into a database and notes the document type. To shrink the number of selectable documents, use <mimeType_sexp> to specify the desired types.

This command also grants permissions to the selected document until the system is restarted. So restart your system and test your app again before publishing.

The default MIME type is "*/*".

Common types are "image/png", "image/jpg", "text/plain" etc.

If a new document directory is needed, use "_Dir". Unfortunately, **Adoc.Open** is not able to select a directory.

Minimum KitKat 4.4 (API 19) is needed.

Example:

```
! As a file browser
startPath$ = "" % Default data path
Adoc.open documentName$, startPath$
File.absolute fileName$,  documentName$
GrabFile result$, fileName$
Print result$
```

or

```
! As a document browser
startPath$ = "" % Default data path
Adoc.open documentName$, startPath$
Adoc.read result$, documentName$
Print result$
```

or

```
! As a document browser perhaps on Google Drive
File.root path$, "_Internal"
absoluteDocsPath$ = "file://" + path$ + "docs.bn"
File.exists ok, absoluteDocsPath$

If ok Then
 Bundle.load mDocs, absoluteDocsPath$
 ! bundle.clear mDocs % If something went wrong at the first tries
 ! In this case do not forget to delete all created Documents also
 ! Document Directories by the Google Drive App before.
 Bundle.contain mDocs, "myDocDirectoryOnGoogleDrive", ex
 If ex Then Bundle.get mDocs, "myDocDirectoryOnGoogleDrive", startPath$
EndIf

If !ok | !ex Then
 rootDir$ = "myDocDirectory"
 Adoc.save startPath$, rootDir$, "", "_Dir"
 Print "startPath$", startPath$
 Bundle.put mDocs, "myDocDirectoryOnGoogleDrive", startPath$
 Bundle.save mDocs, absoluteDocsPath$
EndIf

newDocName$ = "myFirstDocument.txt"
Bundle.contain mDocs, "./" + newDocName$, ex

If ex Then
 message$ = "Take care, because\n you are able to\n  " ~
  + "create a second one\n   with the same name"
 Dialog.message "Document Exists", message$, sel , "SKIP", "OK"
EndIf

If sel <> 1 Then
 Adoc.save documentPath$, newDocName$, startPath$, "text/plain"
 Bundle.put mDocs, "./" + newDocName$, documentPath$
 Bundle.save mDocs, absoluteDocsPath$
EndIf

Adoc.exists ok, documentPath$
If ok Then Adoc.write ws, documentPath$, "My First Text On Google Drive"
Adoc.open documentPath$ ,startPath$
Adoc.read ok, result$, documentPath$ %, "_UTF-8"
Print result$, ok
End
```

```
! Before the third run change your root Document Directory name
! with the Google Drive App.
! You should see, that your Document is still selectable.
! With Adoc.name you can update your Docs bundle.
```

17.9 Adoc.path

Sybtax: Adoc.path <documentPath_svar>, <filePath_sexp>

Returns a document path, beginning with "content://", of the given file path.

If an error occurs, an empty string ("") is returned.

17.10 Adoc.read

Syntax: Adoc.read <success_nvar>, <result_sexp>, <documentPath_sexp>{{, <charSet_sexp>}, <grant_perm_nexp>}

Reads the document specified by <documentPath_sexp> into the string specified by <result_sexp>. Sets <success_nvar> to 1 if no error, or 0 if there was an error.

For binary data, specify the <charSet_sexp> "_ISO-8859-1" instead of the default "_UTF-8".

Required permissions are granted until revoked, or the app is uninstalled.

You can choose between following character sets: `"_US-ASCII"`, `"_UTF-8"`, `"_UTF-16"`, `"_UTF-16BE"`, `"_UTF-16LE"` and `"_ISO-8859-1"`.

Minimum KitKat 4.4 (API 19) is needed.

Use <grant_perm_nexp> to control permission grants. If a Document Provider does not ask for permissions, use 0 (false). Default is 1 (true).

17.11 Adoc.read.file

Syntax: Adoc.read.file <success_nvar>, <filePath_sexp>, <documentPath_sexp>{, <grant_perm_nexp>}

Reads the document specified by <documentPath_sexp> into the file specified by <filePath_sexp>. Sets <success_nvar> to 1 if no error, or 0 if there was an error.

Use <grant_perm_nexp> to control permission grants. If a Document Provider does not ask for permissions, use 0 (false). Default is 1 (true).

17.12 Adoc.rename

Syntax: Adoc.rename <success_nvar>, <documentPath_sexp>, <newName_sexp>

Returns 1 if the document specified by the document path was successfully renamed to <newName_sexp>. Otherwise, 0 is returned.

Minimum Lollipop 5.0 (API 21) is needed.

If the API is 19 or 20, copy the document and delete the first document with **Adoc.delete**.

17.13 Adoc.revoke

Syntax: Adoc.revoke <success_nvar>, <documentPath_sexp>

Returns 1 if the document's access permission specified by the document path was deleted successfully. Otherwise 0 is returned.

Minimum KitKat 4.4 (API 19) is needed.

17.14 Adoc.save

Syntax: Adoc.save <documentPath_svar>, <documentName_svar>{{, <startPath_sexp>}, <mimeType_sexp>}

Opens a system dialog for saving a document, and returns the created or selected path in <documentPath_svar>. If <documentPath_svar> returns an empty string (""), the operation was not successful.

<documentName_svar> specifies the new document name.

An existing start path can be defined by <startPath_sexp>. An empty string tells the document browser to start at the default data path. If this attribute is not used, it starts at the last visited location.

Android stores each document into a database and notes the document type. To shrink the number of selectable documents, use <mimeType_sexp> to specify the desired types.

The default MIME type is "*/*".

Common types are "image/png", "image/jpg",, "text/plain" etc.

If a new document directory is needed, use "_Dir".

In newer Android versions, the document browser also has a menu item to create a new directory.

Minimum KitKat 4.4 (API 19) is needed.

17.15 Adoc.size

Syntax: Adoc.size <documentSize_nvar>, <documentPath_sexp>

Returns the number of bytes in the document identified by the openable document path. Returns -1 if unknown.

17.16 Adoc.write

Syntax: Adoc.write <success_nvar>, <documentPath_sexp>, <newContent_sexp>{{, <charSet_sexp>}, <grant_perm_nexp>}

Returns 1 if the document specified by the document path was successfully filled with the string specified by <newContent_sexp>. Otherwise, 0 is returned.

Note: The specified document must already exist.

For binary data, specify the <charSet_sexp> "_ISO-8859-1" instead of the default "_UTF-8".

Required permissions are granted until revoked, or the app is uninstalled.

You can choose between following character sets: "_US-ASCII", "_UTF-8", "_UTF-16", "_UTF-16BE", "_UTF-16LE" and "_ISO-8859-1".

Minimum KitKat 4.4 (API 19) is needed.

Use <grant_perm_nexp> to control permission grants. If a Document Provider does not ask for

permissions, use 0 (false). Default is 1 (true).

17.17 Adoc.write.file

Syntax: Adoc.write.file <success_nvar>, <documentPath_sexp>, <filePath_sexp>{, <grant_perm_nexp>}

Returns 1 if the document specified by the document path was successfully filled with a file content specified by <filePath_sexp>. Otherwise, 0 is returned.

Note: The specified document must already exist.

Use <grant_perm_nexp> to control permission grants. If a Document Provider does not ask for permissions, use 0 (false). Default is 1 (true).

18 Email Commands

18.1 Email.send

Syntax: Email.send <recipient_sexp>|Array$[], <subject_sexp>, <body_sexp>{{{{{, <sendVia_sexp>}, <cC_sexp>|Array$[]}, <bCC_sexp>|Array$[]}, <attachment_svar>|Array$[]}, <fileType_svar>}

The email message in the Body string expression will be sent to the named recipient with the named subject heading.

The email message in the **<body_sexp>** body string expression will be sent to the named recipient(s) with the named subject heading. If **<recipient_sexp>** is "_Send", the recipient has to be defined within the messenger app.

<sendVia_sexp> specifies an app to perform the send, such as Gmail, WhatsApp, DropBox or OneDrive, etc. See also **App.installed**. If **<sendVia_sexp>** is an empty string, an app picker with all possible apps pops up. Maybe also a PDF reader for a single file.

<cC_sexp> and **<bCC_sexp>** place the CC and BCC recipient(s) in the mail header.
<attachment_svar> specifies the attachment(s). For a single file only a single string variable is accepted.

The command expects double quotes for undefined arguments.

Files from the _Internal file folder have to be copied into an external folder before sending.

In case of Android 10+ with Scoped Storage, you have to copy your files into the File Provider directory.

Example:

```
! Get the File Provider directory path.
File.root fpPath$, "_FileProvider"
! Copy your files.
File.copy "whee.mp3", fpPath$ + "/" + "whee.mp3", "_ReplaceExisting"
! Set the permissions for reading.
Bundle.put pB, "_FileP_Read", 1
Provider pB
...
Program.info bInf
Bundle.get bInf, "_PackageName", pn$
fpFromOutside$ = "content://" + pn$ + "/" + "whee.mp3" % Path within the
                                                       % File Provider dir.
Emal.send ..., fpFromOutside$
...
! Withdraw the authorization.
! Note calling the email app take some time.
! Maybe use Timer.set to prevent closing before the file(s) are taken over.
Bundle.put pB, "_FileP_Read", 0
Provider pB
! Usually delete the copied files after sending
File.delete dOk, fpPath$ + "/" + "whee.mp3"
```

See also Provider, File.copy, File.Move, File.Delete

Known issues:

- Gmail: Is not able to read the file size or the file itself.
- Signal: Is not able to read from the FileProvider because of calling permissions.

Some email app IDs:

- Gmail: com.google.android.gm
- Outlook: com.microsoft.office.outlook
- WEB.de: de.web.mobile.android.mail
- Signal: org.thoughtcrime.securesms.sharing.v2.ShareActivity

Example:

```
gmail$ = "com.google.android.gm"
Array.load cC$[], "bob.ex@example.com", "harry.c@examples.com"
FILE.ROOT dataPath$
Array.load files$[], "file://" + dataPath$ + "/" + "cartman.png", ~
 "file://" + dataPath$ + "/" + "whee.mp3"
Email.send "b.ex@gmail.com", "Mail Subject", "Message", gmail$, cC$[], ~
 "", files$[]
```

19 Files and Paths

Android devices may have several file storage devices. BASIC! uses one of these devices as its *base drive*. You can select a different base drive in the *Menu->Preferences* item Base Drive. In this manual, the notation **<pref base drive>** refers to the base drive you selected in Preferences.

BASIC! can work with files anywhere on the base drive, but most file operations are done in BASIC!'s *base directory*. Except when you create a standalone apk file, the base directory is **<pref base drive>/rfo-basic**. All file paths are relative to a subdirectory of the base directory.

19.1 Paths Explained

A path describes where a file or directory is located relative to another directory.

A file is a container for data. A directory is a container for files and other directories. This container is called a "directory" because it is a listing of the items it contains. You can also call it a "folder", and you can say "a folder is a container for files and other folders." Both expressions mean the same thing. A directory that is in a directory may have other directories in it, and those directories may contain other directories, and so on. This results in a "tree" of directories and files, called a *file system*. A file system organizes data on a storage device, such as a disk or a memory card.

Absolute paths: A path that is relative to the root directory is called an *absolute path*. For example, the absolute path to pictures you take with an Android camera may be "/sdcard/DCIM/Camera". In BASIC!, the *base directory* is not a *root directory*, so *BASIC! does not use absolute paths*.

Relative paths: A path that is relative to anything except the root directory is called a *relative path*. Starting from "/sdcard", the relative path down to Camera is "DCIM/Camera". Use the "../" path notation to go back up: from Camera, the path to "/sdcard/DCIM" is ".." and to "/sdcard" is "../.." .

The relative path from "/sdcard/DCIM/Camera" to "/sdcard/DCIM/.thumbnails" is "../.thumbnails". With this notation, you can reach any file in the file system.

All paths in BASIC! are relative to a default path that depends on the kind of data you want to use. These default paths are explained in the next section.

19.2 Paths in BASIC!

BASIC! files are stored in subdirectories of the base directory, "<pref base drive>/rfo-basic/". Files are grouped by type, as follows:

- BASIC! program files are in **rfo-basic/source/**
- BASIC! data files are in **rfo-basic/data/**
- BASIC! SQLite databases are in **rfo-basic/databases/**

All of the BASIC! file commands assume a certain default path. The default path depends on the type of file each command expects to handle:

- The **Include** and **Run** commands expect to load program files, so they look in **rfo-basic/source/**.
- **SQLite** operations look for database files in **rfo-basic/databases/**.
- All other file operations look for data files in **rfo-basic/data/**.

If you give a filename to a file command, it looks for that filename in the default directory for commands of that type. If you want to work with a file that is not in that directory, you must specify a relative path to the appropriate directory. BASIC! adds your path to the default path.

- To read the file "names.txt" in "rfo-basic/data/", the path is "names.txt".

- To read the program file "sines.bas" in "rfo-basic/source", the path is "../source/sines.bas".

19.3 Paths Outside of BASIC!

BASIC! runs on an Android device, and its files are part of the Android file system.

You can use relative paths to access files outside of the base directory. To continue the example from the previous section, assume the absolute Android path to the <pref base drive> is "/sdcard":

- To read the music file "rain.mp3" in "/sdcard/music/", the BASIC! path is "../../music/rain.mp3". The path is relative to the default directory for general data "<pref base drive>/rfo-basic/data/".

Use care when writing paths that look up from the <pref base drive>. The directory name may not be what you expect, and it may not be the same on all Android devices.

Every file on an Android device has an absolute path. Unfortunately, on most Android devices every file has several absolute paths. The default <pref base drive> can be reached with the absolute Android path "/sdcard". The name "sdcard" is a shortcut (a *symbolic link*) that means different things on different devices. "<pref base drive>/rfo-basic/../" may not get to the Android directory "/sdcard", but to something like "/storage/emulated/legacy".

You can use the command **File.root** to get the absolute Android path to the BASIC! <pref base drive>.

19.4 Paths and Case-sensitivity

While the Android file system is normally case-sensitive. However, the FAT file system, often used on SD cards, memory sticks, etc., is case-insensitive. When handling files on these devices, Android – and therefore BASIC! – can not differentiate between names that differ only in case. In your BASIC! program, the two paths "../../music/rain.mp3" and "../../MUSIC/Rain.MP3" will both access the same file.

The rules change if you compile the same BASIC! program into a standalone apk. The file system inside the apk is case-sensitive. The paths "../../music/rain.mp3" and"../../MUSIC/Rain.MP3" access different files. If the actual path in your build project is "Assets/<project>/MUSIC/Rain.MP3", then using the second path would succeed, but the first path would fail.

To prevent any error, it is good practice to match case exactly in file paths and names.

19.5 Mark and Mark Limit

Note: This is an advanced file management technique that you will rarely need to use. However, see the note below about Out of Memory errors when reading very large files.

Every file has a mark and a mark limit. The mark is a position, and the mark limit is a size. As you read a file, its data is copied into a buffer. The buffer starts at the mark, and its length is the mark limit. BASIC! uses the buffer to allow you to reposition within the file. You are really repositioning within the buffer.

If you do not mark a file, then the first time you read it or set a position in it, the mark is set at position 1 and the mark limit is set to the size of the file. This allows you to position and read anywhere in the file.

The **Text.position.mark** and **Byte.position.mark** commands override the default mark and mark limit. You can change the mark and mark limit as often as you like, but there is only one mark in a file.

You cannot set a position before the mark. If you try, the file will be positioned at the mark. You will not be notified that the current position is different from what you requested, but you can use **Text/Byte.position.get** to determine the real position. Since the default mark position is 1, you can position anywhere if you never set a mark.

You cannot make the buffer smaller. If you want the buffer to be smaller than the file, you must execute **Text/Byte.position.mark** before reading the file or setting a position. The smallest buffer size available is 8096 bytes, but it is not an error to specify a smaller number. Since the default mark limit is the file size, you can position anywhere if you never set a mark limit.

If you read or position past the end of the buffer (more than **marklimit** bytes beyond the mark), the mark is invalid. It is an error to try to move the position backward when the mark is invalid. You will see the error message "**Invalid mark**". Since the default buffer is the whole file, you will never see this error if you never set a mark.

There is only one condition that requires you to use this command. **If you open a very large file, the default buffer size may be too large.** Reading or positioning to the end of the file may cause an **Out Of Memory** error. To avoid this error, your must use **Text.position.mark** to reduce the buffer size.

19.6 Files and Resources

If your program is compiled into an APK, file handling can be a little more complicated. The APK has several internal directories, but the two of interest to us are the **assets** and **res** directories.

Standard BASIC! loads its sample programs and the data files they need from the **assets** directory of the APK. Android treats the **assets** directory like a file system. At startup, BASIC! simply copies the entire **assets/rfo-basic** directory to the SD card.

Your application can use the **assets** directory, too. Most of the BASIC! file-handling commands look in the SD card file system first. If the file does not exist on the SD card, your program compiled into an APK looks in your projects' **res** directory, and finally in the **assets** directory. Resources may be built into the APK in either the **res** or **assets** directory, but **res** is not treated as a file system and has more restrictive naming rules. For example, the name of a resource in **res** must not contain spaces or hyphens.

If a directory was empty at compiling, it will not be created under assets. That is why there are no empty directories in assets.

Both **Byte.open** and **Text.open** are able to open resources in your APK. However, Zip.open cannot open resources; it looks only in the SD card file system. If you want to read a ZIP that is in **assets**, you must first copy it from **assets** to the SD card. Then you can open the SD card file with Zip.Open.

File.Exists looks only for files on the SD card, but **File.Type** works with items in **assets** as well. You can use them together to determine where an item is. The other **File.*** commands work with either files or resources. **Font.Load** can load a resource if it does not find the font file on the SD card.

Files in **assets** are read-only. Your program can create and modify files on the SD card. It cannot create or modify files in **assets**.

File names in **assets** are case-sensitive. If your program looks for a file on the SD card, the name is not case-sensitive: "meow.wav" and "Meow.WAV" are the same file. However, to find a file in **assets**, your program must name the file exactly as you put it in **assets**. **Audio.Load aft, "meow.wav"** will not find **assets/rfo-cats/data/Meow.WAV**.

19.7 File Closing

It is essential to close an output file before your program ends. This should also be done before the RUN command is executed.

20 File Commands

Note: It is essential to close an output file before your program ends. This should also be done before the RUN command is executed.

20.1 Dir

See **File.dir**

20.2 File.absolute

Syntax: File.absolute <absolute_svar>, <path_sexp>

Returns the absolute file path of <path_sexp> in <absolute_svar>.

Tries to convert a document path beginning with "content://" into an absolute file path. Minimum KitKat 4.4 (API 19) is needed. If ":open uri with ADOC.Read or ADOC.Write" is returned, use these **ADOC** commands for access.

20.3 File.copy

Syntax: File.copy <sourcePath_sexp>, <targetPath_sexp>{, <modes_sexp>}

Copies a File or Directory.

By default, the copy fails if the target file already exists. Use **File.exists** first to prevent this condition.

Directories can be copied. However, files inside the directory are not copied, so the new directory is empty even if the original directory contains files.

When copying a symbolic link, the target of the link is copied. If you want to copy the link itself, and not the contents of the link, specify the `_ReplaceExisting` mode.

Additionally, the following modes are supported:

`_ReplaceExisting` – Performs the copy even if the target file already exists. If the target is a symbolic link, the link itself is copied (and not the target of the link). If the target is a non-empty directory, the copy fails with the `FileAlreadyExistsException` error.

`_CopyAttributes` – Copies the file attributes associated with the file to the target file. The exact file attributes supported are file system and platform dependent, but last-modified-time is supported across platforms and is copied to the target file.

To specify more than one option, put a comma separated string into <modes_sexp>, such as `"_ReplaceExisting, _CopyAttributes"`.

This command is only supported beginning with Android 8+ (API 26+).

20.4 File.delete

Syntax: File.delete <lvar>, <path_sexp> {, <recursive_nexp>}

The file or directory at <path_sexp> will be deleted, if it exists. If the file or directory did not exist before the delete, or it could not be deleted, the <lvar> will contain zero. If the file or directory did exist and was deleted, the <lvar> will be returned as not zero.

The default path is "<pref base drive>/rfo-basic/data/".

If <recursive_nexp> is = 1 all sub directories and files are deleted as well. If <recursive_nexp> is = 2 the directory specified by <path_sexp> is deleted as well. Default is 0.

If <path_sexp> contains more than one directory level, as in "dir1/dir2/file1", this command will try to delete only the last item. So in the above example, only "file1" would be deleted while the folders "dir1" and "dir2" will not be deleted.

The following examples assume that "blabla" is a file:

```
File.delete L, "blabla"      % deletes file "blabla" in directory
                             % <pref base drive>/rfo-basic/data/

File.delete L, "dir1/blabla" % deletes file "blabla" in directory
                             % <pref base drive>/rfo-basic/data/dir1/
```

The following examples assume that "blabla" is a directory:

```
File.delete L, "blabla"      % deletes directory "blabla" from directory
                             % <pref base drive>/rfo-basic/data/
                             % but only if "blabla" is empty.

File.delete L, "dir1/blabla" % deletes directory "blabla" from directory
                             % <pref base drive>/rfo-basic/data/dir1/
                             % but only if "blabla" is empty.
```

20.5 File.dir

Syntax: File.dir <path_sexp>, Array$[] {{{{, <dirmark_sexp>}, <timeStamp_nexp>}, <recursive_nexp>}, <type_sexp>}

Returns the names of the files and directories in the path specified by <path_sexp>. The path is relative to "<pref base drive>/rfo-basic/data/". **Dir** is a valid alias for this command.

The names are placed into Array$[]. The array is sorted alphabetically with the directories at the top of the list. If the array exists, it is overwritten, otherwise a new array is created. The result is always a one-dimensional array.

A directory is identified by a marker appended to its name. The default marker is the string "(d)". You can change the marker with the optional directory mark parameter <dirmark_sexp>. If you do not want directories to be marked, set <dirmark_sexp> to an empty string, "".

If <path_sexp> starts with "**asset://**", you will get the directories and files from the **APK** assets. Depending on the structure of the assets, the distinction between directory or file is only possible via the point. In this case, file time stamps are not available.

Keep also in mind, that there are **no empty directories** in assets!

If the directory is empty, **File.dir** returns an array with one item containing a string with one space (" ") in it.

Options of <timeStamp_nexp>:

- 0 no time stamp (default)
- 1 with time stamp as time in milliseconds + ":" + file name, but unsorted
- 2 with time stamp as time in milliseconds + ":" + file name, sorted in ascending order
- 3 with time stamp as time in milliseconds + ":" + file name, sorted in descending order

If <recursive_nexp> is > 0 all sub directories are searched as well. Default is 0. If only files or directories are needed, use <type_sexp> with "_F" for files and "_D" for directories. Default is "_DF".

Example:

```
File.root path$, "_Mnt"
File.dir path$, dirArray$[], "", 0 , 1 , "_F"
List.create s, dirItems
List.add.array dirItems, dirArray$[]
List.create s, dirFilteredItems
List.match,  dirFilteredItems, dirItems, "png",,,, "_Ends_With_IgnoreCase"
Debug.on
Debug.dump.list dirFilteredItems
```

20.6 File.encoding

Syntax: File.encoding <enc_svar>, <path_sexp>

Returns the character set encoding of a file.

20.7 File.exists

Syntax: File.exists <lvar>, <path_sexp>

Reports if the <path_sexp> directory or file exists. If the directory or file does not exist, <lvar> will contain zero. If the file or directory does exist, <lvar> will be returned as not zero. If the file or directory is readable, <lvar> returns 2.0. If the file or directory is writeable, <lvar> returns 3.0. If the file or directory is readable and writeable, <lvar> returns 4.0.

The default path is "<pref base drive>/rfo-basic/data/".

<path_sexp> can be an URI which starts with "file://".

For a file path like “/external/images/media/556”, use "file://" + “/external/images/media/556” so that **File.exists** will be able to handle it. In this case, <path_sexp> returns a new absolute file path, but only if <path_sexp> is a single value like fn$ and not a simple string expression.

File.exists has no access to assets and resources in APKs.

20.8 File.lastModified

Syntax: File.lastModified <nvar>, <path_sexp>

Returns the time when this file was last modified, measured in milliseconds since January 1st, 1970, midnight GMT.

If there was an error, such as trying to access a Resource or Asset file, <nvar> returns 1.

Example:

```
File.root pp$ ,"_SourceSamples"
File.lastModified lm, "file://"+ pp$ + "/" + "f01_commands.bas"
? Using$("", "%TF %tT", lm, lm) % Returns "2019-06-30 21:01:01"
File.set. lastModified lm, "file://"+ pp$ + "/" + "f01_commands.bas", lm + 6000
? Using$("", "%TF %tT", lm, lm) % Returns "2019-06-30 21:01:07"
```

20.9 File.md5

Syntax: File.md5 <svar>, <path_sexp>

Returns the MD5 hash of the file specified by <path_sexp>.

It is only useful if a compare of the contents of a file is needed, because this algorithm is not secure but faster.

20.10 File.mkDir

Syntax: File.mkDir <path_sexp>

Before you can use a directory, the directory must exist. Use this command to create a directory named by the path string <path_sexp>. **MkDir** is a valid alias for this command.

The new directory is created relative to the default directory "<pref base drive>/rfo-basic/data/". For example:

- To create a new directory, "homes", in "<pref base drive>/rfo_basic/data/", use the path "homes/", or simply "homes".
- To create a new directory, "icons", in the root directory of the SD card, use "../../icons".

If <path_sexp> contains more than one directory level, all levels will be created. For example:

```
file.makedir "one/two"
```

will create "<pref base drive>/rfo_basic/data/one/two".

20.11 File.move

Syntax: File.move <sourcePath_sexp>, <targetPath_sexp>{, <modes_sexp>}

Moves a File or Directory.

By default, the copy fails if the target file already exists. Use **File.exists** first to prevent this condition.

Empty directories can be moved. If the directory is not empty, the move is allowed when the directory can be moved without moving the contents of that directory. On the Android system, moving a directory within the same partition generally consists of renaming the directory. In that situation, this method works even when the directory contains files.

Additionally, the following modes are supported:

`_ReplaceExisting` – Performs the move even when the target file already exists. If the target is a symbolic link, the symbolic link is replaced but what it points to is not affected.

`_AtomicMove` – Performs the move as an atomic file operation. If the file system does not support an atomic move, an exception is thrown. With an _AtomicMove, you can move a file into a directory and be guaranteed that any process watching the directory accesses a complete file.

To specify more than one mode, put a comma separated string into <modes_sexp> like this `"_ReplaceExisting, _AtomicMove"`.

This command is only supported beginning with Android 8+ (API 26+).

20.12 File.reader

Syntax: File.reader <result_svar>, <path_sexp>{, <unicode_flag_lexp>|<charset_sexp>}

Copies the entire contents of the file at <path_sexp> to the string variable <result_svar>. By default, **File.reader** assumes that the file contains binary bytes or ASCII characters. If the optional <unicode_flag_lexp> evaluates to true (a non-zero numeric value), **File.reader** can read Unicode text.

Instead of <unicode_flag_lexp>, use <charset_sexp> to choose between the following character sets: `"_US-ASCII"(1), "_UTF-8"(1), "_UTF-16", "_UTF-16BE", "_UTF-16LE"` and `"_ISO-8859-1"(0)`.

If the file does not exist or cannot be opened, the <result_svar> is set to an empty string and you can use the **GETERROR$()** function to get more information. If the file is empty, the <result_svar> is an empty string, but **GETERROR$()** returns "No error".

For text files, either ASCII or Unicode, the **Split** command can be used to split the <result_svar> into an array of lines. **File.reader** can also be used grab the contents of a text file for direct use with **Text.input**:

```
File.reader text$, "MyJournal.txt"
Text.input EditedText$, text$
```

20.13 File.rename

Syntax: File.rename <old_path_sexp>, <new_path_sexp>

The file or directory at old_path is renamed to new_path. If there is already a file present named <new_path_sexp>, it is silently replaced. **Rename** is a valid alias for this command.

The default path is "<pref base drive>/rfo-basic/data/".

The rename operation can not only change the name of a file or a directory, it can also move the file or directory to another directory.

For example:

```
File.rename "../../testfile.txt", "testfile1.txt"
```

removes the file, testfile.txt, from "<pref base drive>/", places it into "sdcard/rfo-basic/data/" and also renames it to testfile1.txt.

20.14 File.replace

Syntax: File.replace <startPath_sexp>, <replace_list_nexp>

Replaces a string in a file or files within directories.

This will start at a location, and if that location is a file it will process it. If the location is a directory it will process all of the files in the directory. If the directory contains other directories, it will continue to recursively process until all files in all sub directories of the original location have been processed.

The entries of a string list, pointed at by <replace_list_nexp>, must be in order of replace type, the string to be replaced and the replacement. This order can be used multiple times to replace more than one text string. It is replaced in turn.

Possible replace types are:

`_Replace` - Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence. The placement proceeds from the beginning of the string to the end. For example, replacing "aa" with "b" in the string "aaa" will result in"ba" rather than "ab".

`_ReplaceIgnoreCase` - Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence. The placement proceeds from the beginning of the string to the end. For example, replacing "aa" with "b" in the string "Aaa" will result in"ba" rather than "ab".

`_ReplaceAll` - Replaces each substring of this string that matches the given regular expression with the given replacement.

`_ReplaceFirst` - Replaces the first substring of this string that matches the given regular expression with the given replacement.

Example:

```
List.create s, rpl
List.add rpl, "_ReplaceFirst", "m.c", "g.c", "_Replace", "good", "bad"
File.replace "testDir", rpl
```

20.15 File.root

Syntax: File.root <full_path_svar>{, <dirType_sexp>}

Returns the canonical path from the file system root to "<pref base drive>/rfo-basic/data", the default data directory, in <svar>. The <pref base drive> is expanded to the full absolute Android path from the file system root, "/".

You cannot use this path in any BASIC! command, as BASIC! paths are relative to a command-dependent default directory. However, you can use it to compute a relative path to parts of the Android file system outside of the BASIC! <pref base drive>.

The system constants in **<dirType_sexp>** enables easy access to BASIC! and other folders in conjunction to **Scoped Storage** (Android (10+) 11+):

- _Alarms
- _App
- _AppPath
- _Asset_Cache
- _BasicSystem
- _Bluetooth
- _Cache
- _Data
- _Database
- _Dcim
- _Documents
- _Downloads
- _External
- _Internal
- _InternalOnExternal
- _InternalOnSdRemovable*
- _Mnt
- _Movies
- _Music
- _Notifications
- _Pictures
- _Podcasts
- _ProgramPath
- _Ringtones
- _SdRemovable*
- _Service
- _Source
- _SourceSamples
- _ScreenShots
- _Storage
- _System

* Fully available with Android 4.4 KitKat (API19) and later. For older APIs, the interpreter searches for sdcard1, sdcard2, extSdcard and sd-ext.

If an `_InternalOnExternal` or `_InternalOnSdRemovable` folder did not exist, it will be created. The folder _Documents is not automatically created on some devices, but you can do it yourself with **File.mkdir**.

If you use the internal cache folder (`_Cache`), its files will be the first to be deleted if the device runs low on storage. There is no indication when these files will be deleted. Note: you should not rely on the system deleting these files for you; you should always have a reasonable maximum, such as 1 MB, for the amount of space you consume with cache files, and prune those files when exceeding that space.

The _Asset_Cache folder, as part of the _Cache folder, will be created automatically. For example: Byte.open myFileTable, "asset://" + myFileNamePath$ if the folder does not exists.

Note: **"file://"** + <full_path_svar> is required if you want to use it directly in other BASIC! commands.

Example:

```
File.root dataPath$, "_Documents"
fn$ = "file://" + dataPath$
File.exists ok, fn$
If ok = 0 Then
 File.root newFolder$, "_External"
 newFolder$ = "file://" + newFolder$ + "/Documents"
 File.mkdir newFolder$
EndIf
```

Rfo-Basic ignores the slash at the beginning, because it does not need it. If you want to use an absolute path in OliBasic, your path$ should begin with "file://" + ...

In OliBasic, a slash at the path beginning has the same function as "file://". Thus, **File.root mPath$** returns an absolute path, which can be used directly like this: mPath$ = mPath$ + "/" + "cartman.png". If you get in trouble, use "file://" for absolute file or directory paths.

20.16 File.root.reset

Syntax: File.root.reset

Resets the default data and database paths to what they were when the program started.

20.17 File.root.set.data

Syntax: File.root.set.data <check_svar>, <root_path_sexp>

Sets the default data path to a different location by the parameter <root_path_sexp>.

In every case, relative paths of <root_path_sexp> start at (Base Directory)/rfo-basic/data. Absolute paths are also possible. <check_svar> returns the absolute path. If an error occurs, <check_svar> begins with "Error:".

This command enables to structure the data of different programs.

It is also possible to store your data directly into the Adoc Provider or File Provider to share the program data with other apps.

Example:

```
! The program name is myTest.bas. You can create your default data path by:
File.root.set.data ok$, "../projects/myTest"
```

```
Print ok$ % Returns .../rfo-basic/projects/myTest/data
```

20.18 File.root.set.databases

Syntax: File.root.set.databases <check_svar>, <root_path_sexp>

Sets the default database path to a different location by the parameter <root_path_sexp>.

In every case, relative paths of <root_path_sexp> start at (Base Directory)/rfo-basic/**data**. Absolute paths are also possible. <check_svar> returns the absolute path. If an error occurs, <check_svar> begins with "Error:".

This command enables to structure the databases of different programs.

It is also possible to store your databases directly into the Adoc Provider or File Provider to share the program databases with other apps.

Example:

```
! The program name is myTest.bas. You can create your default database path by:
File.root targetPath$, "_FileProvider"
File.root.set.databases ok$, targetPath$
Print ok$ % Returns /data/data/com.my.app/fileProvider/databases

Bundle.put bnd, "_FileP_Read", 1
Bundle.put bnd, "_FileP_Write", 1
Provider bnd % Now the databases are accessible also from outside.
```

20.19 File.select

Syntax: File.select <filePath_svar>, <startPath_sexp>{, <settings_bundle_nexp>}

Returns a chosen directory or file path in <filePath_svar>. The selection process begins at the path specified by <startPath_sexp>.

If you use buttons, the result is different. See the table below.

The optional layout bundle <layout_bundle_nexp> controls the **File.select** layout:

Settings Bundle		
Key	**Possible Values**	**Description**
_Style	_Default	Default theme
	_Dark	Dark theme and Autosize mode
	_Bright	Bright theme and Autosize mode
_Icon	String	File path of a header icon
_Action	String	Action as start of the title.
_PositionH	_Left	
	_Center	Default
	_Right	
_PositionV	_Top	
	_Center	Default
	_Bottom	
_DisplayPath	numeric	If 0, no absolute path will be displayed in the title. If > 0, the absolute path will be displayed. (Default)

Settings Bundle		
Key	**Possible Values**	**Description**
_EndsWith	String like "?/?/? …"	If the string is not empty, only files with given endings will be returned. A string like "jpg/gif/mp3 …" with endings like ". jpg", ". gif", ".mp3" … will only return files with these endings. The delimiter is the slash "/", because the comma can be part of a file name. If the string is "-1", only directories will be returned.
_UpperStart	numeric	If 0, no path upper the start path will be displayed. (Default) If > 0, paths upper the start path will be displayed.
_OnlyDirs	numeric	If > 0, only directories are displayed. The default is 0.
_Button1	Text of Button 1	**Positive** button. Default is "⚭ ⚭". Pressing returns the next upper directory if possible.
_Button1Size	numeric	
_Button1Color	{Alpha, } Red, Green, Blue (comma delimited string) or _{Alpha, } ColorName ({comma delimited } string) or #{hn}hnhnhn (hex string)	
_Button2	Text of Button 2	**Neutral** button. Default is "". You could use "+ " + CHR$(128194) (🗀) as an example. Pressing returns a "+" and the current absolute path follows.
_Button2Size	numeric	
_Button2Color	{Alpha, } Red, Green, Blue (comma delimited string) or _{Alpha, } ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_Button3	Text of Button 3	**Negative** button. Default is "X". Pressing returns a "-" and the current absolute path follows.
_Button3Size	numeric	

Settings Bundle		
Key	**Possible Values**	**Description**
_Button3Color	{Alpha, } Red, Green, Blue (comma delimited string) or _{Alpha, } ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_Cancelable	numeric	If > 0 the dialog is cancelable. Default is 1.

20.20 File.set.lastModified

Syntax: File.set.lastModified <nvar>, <path_sexp>, <new_ time_nexp>

The parameter <new_ time_nexp> sets the new time for the last modified time. The time is measured in milliseconds since January 1st, 1970, midnight GMT.

As a feedback, <nvar> returns the new measured time.

If there was an error, such as trying to access a Resource or Asset file, <nvar> returns 1.

20.21 File.sha

Syntax: File.sha <svar>, <path_sexp>{, <algorithm_sexp>}

Returns the SHA hash of the file specified by <path_sexp> and the algorithm <algorithm_sexp>. Possible algorithms are _SHA-1, _SHA-224, _SHA-256, _SHA-384, _SHA-512, _SHA-512/224 or _SHA-512/256. Default is _SHA-256.

Secure hash algorithms – SHA-1(insecure!), SHA-224, SHA-256, SHA-384, SHA-512 – are used for computing a condensed representation of electronic data (message). When a message of any length less than 2^{64} bits (for SHA-1, SHA-224, and SHA-256) or less than 2^{128} (for SHA-384 and SHA-512) is input to a hash algorithm, the result is an output called a message digest. A message digest ranges in length from 160 to 512 bits, depending on the algorithm.

20.22 File.size

Syntax: File.size <size_nvar>, <path_sexp>

The size, in bytes, of the file at <path_sexp> is returned in <size_nvar>. If there is no file at <path_sexp>, this command generates a run-time error.

The <path_sexp> is appended to the default path, "<pref base drive>/rfo-basic/data/".

20.23 File.type

Syntax: File.type <type_svar>, <path_sexp>

Returns a one-character type indicator in <type_svar> for the file at <path_sexp>. The <path_sexp> is appended to the default data path, "<pref base drive>/rfo-basic/data/". The type indicator values are:

Indicator:	Meaning:
"d"	"directory" – path names a directory
"f"	"file" – path names a regular file
"o"	"other" – path names a special file
"x"	file does not exist

20.24 File.writer

Syntax: File.writer <path_sexp>, <string_sexp>{, <charset_sexp>}

Writes the contents of a string expression to a file.

Use the optional <charset_sexp> to choose between the following character sets: "_US-ASCII", "_UTF-8"(default), "_UTF-16", "_UTF-16BE", "_UTF-16LE" and "_ISO-8859-1".

20.25 GrabFile

Syntax: GrabFile <result_svar>, <path_sexp>{, <unicode_flag_lexp>|<charset_sexp>}

Copies the entire contents of the file at <path_sexp> to the string variable <result_svar>. By default, **GrabFile** assumes that the file contains binary bytes or ASCII characters. If the optional <unicode_flag_lexp> evaluates to true (a non-zero numeric value), **GrabFile** can read Unicode text.

Use <charset_sexp> instead of <unicode_flag_lexp> to choose between the following character sets: "_US-ASCII"(1), "_UTF-8"(1), "_UTF-16", "_UTF-16BE", "_UTF-16LE" and "_ISO-8859-1"(0).

If the file does not exist or cannot be opened, the <result_svar> is set to the empty string, "", and you can use the **GetError$()** function to get more information. If the file is empty, the <result_svar> is an empty string, "", but **GetError$()** returns "No error".

For text files, either ASCII or Unicode, the **Split** command can be used to split the <result_svar> into an array of lines. **GrabFile** can also be used grab the contents of a text file for direct use with **Text.input**:

```
GrabFile text$, "MyJournal.txt"
Text.input EditedText$, text$
```

20.26 GrabURL

Syntax: GrabURL <result_svar>, <url_sexp>{{, <timeout_nexp>},<unicode_flag_lexp>|<charset_sexp>}

Copies the entire source text of the URL <url_sexp> to the string variable <result_svar>. The URL may specify an Internet resource or a local file. Cached files will be ignored. If the URL does not exist or the data cannot be read, the <result_svar> is set to the empty string, "", and you can use the **GetError$()** function to get more information.

If the optional <timeout_nexp> parameter is non-zero, it specifies a timeout in milliseconds. Thus 0 specifies no time-out. This is meaningful only if the URL names a resource on a remote host. If the timeout time elapses and host does not connect or does not return any data, **GetError$** reports a socket timeout.

By default, **GrabURL** assumes that the file contains Unicode text. If the optional <unicode_flag_lexp> evaluates to false (a zero numeric value), **GrabURL** can read binary bytes or ASCII characters.

Note: The **default** <unicode_flag_lexp> setting is the opposite in **GrabFile**!

Use <charset_sexp> instead of <unicode_flag_lexp> to choose between the following character sets: "_US-ASCII"(1), "_UTF-8"(1), "_UTF-16", "_UTF-16BE", "_UTF-16LE" and "_ISO-8859-1"(0).

If the named resource is empty, the <result_svar> is empty, "", and **GetError$()** returns "No error".

The **Split** command can be used to split the <result_svar> into an array of lines for ASCII or Unicode text.

20.27 MkDir

See **File.mkDir**.

20.28 Rename

See **File.rename**.

21 File Byte Commands

Byte file I/O can be used to read and write any type of file (.txt, .jpg, .pdf, .mp3, etc.). Each command reads or writes one byte or a sequence of bytes as binary data.

Note: It is essential to close an output file before your program ends. This should also be done before the RUN command is executed.

21.1 Byte.close

Syntax: Byte.close <file_table_nexp>

Closes the previously opened file.

21.2 Byte.copy

Sytax: Byte.copy <file_table_nexp>, <output_file_sexp> {{, <append_nexp>}, <break_nexp>}

Copies the previously open input file represented by <file_table_nexp> to the file whose path is specified by <output_file_sexp>. The default path is "<pref base drive>/rfo-basic/data/".

If <file_table_nexp> = -1 then a run-time error will be thrown.

All bytes from the current position of the input file to its end are copied to the output file. Both files are then closed.

If you have read from the input file, and you want to copy the whole file, you must reset the file position to 0 with **Byte.position.set**. However, if you have changed the file mark with **Byte.position.mark**, or if you reading a non-local (internet) file, you can't reset the file position to 0. Instead, you must close and reopen the file.

You should use **Byte.copy** if you are using Byte I/O for the sole purpose of copying. It is thousands (literally) of times faster than using **Byte.read**/**Byte.write**.

If you want to append the input file content to the output file, the optional <append_nexp> parameter has to be > 0. Default is 0.

If the output file does not exist, a new file will be created before writing.

If <break_nexp> is greater than zero, **Byte.copy** can be interrupted by an interrupt.

21.3 Byte.eof

Syntax: Byte.eof <file_table_nexp>, <lvar>

Reports an opened file's end-of-file status. If the file is at EOF, the <lvar> is set true (non-zero). If the file or directory is not at EOF, the <lvar> is set false (zero).

A file opened for write or append is always at EOF. A file opened for read is not at EOF until all of the data has been read and then one more read is attempted. That read will have returned the value -1. **Byte.position.set** may also position the file at EOF.

21.4 Byte.open

Syntax: Byte.open {r|w|a}, <file_table_nvar>, <path_sexp>

The file specified by the path string expression <path_sexp> is opened. If the path is a URL starting with "http…" then an Internet file is opened. Otherwise, the <path_sexp> string is appended to the

default path "<pref base drive>/rfo-basic/data/".

If the URL starts with "asset://" + myFileNamePath$, the _Asset_Cache folder as part of the _Cache folder will be created if the folder does not exists. So a later Byte.copy or other command will be able to access the file from the App's Asset Cache Directory. Larger files should be deleted after use.

The first parameter is a single character that sets the I/O mode for this file:

Parameter	Mode	Notes
r	read	File exists: Reads from the start of the file. File does not exist: Error (see below).
w	write	File exists: Writes from the start of the file. Writes over any existing data. File does not exist: Creates a new file. Writes from the start of the file.
a	append	File exists: Writing starts after the last line in the file. File does not exist: Creates a new file. Writes from the start of the file.

A file table number is placed into the numeric variable <file_table_nvar>. This value is for use in subsequent **Byte.read.***, **Byte.write**.*, **Byte.eof**, **Byte.position.***, **Byte.truncate**, **Byte.copy**, or **Byte.close** commands.

If a file opened for read does not exist then <file_table_nvar> will be set to -1. The program can check for this and either create the file or report the error to the user, possibly using the **GetError$()** function.

21.5 Byte.position.get

Syntax: Byte.position.get <file_table_nexp>, <position_nvar>

Gets the position of the next byte to be read or written. The position of the first byte is 1. The position value will be incremented by 1 for each byte read or written.

The position information can be used for setting up random file data access.

If the file is opened for append, the position returned will be the length of the file plus one.

21.6 Byte.position.mark

Syntax: Byte.position.mark {{<file_table_nexp>}{, <marklimit_nexp>}}

Marks the current position in the file, and sets the mark limit to <marklimit_nexp> bytes.

Both parameters are optional. If the file table index <file_table_nexp> is omitted, the default file is the last file opened; you must ensure that the last file opened was a **Byte** file opened for reading. If the mark limit <marklimit_exp> is omitted, the default value is the file's current mark limit.

Please read **Files and Paths → Mark and Mark Limit**.

21.7 Byte.position.set

Syntax: Byte.position.set <file_table_nexp>, <position_nexp>

Sets the position of the next byte to be read from the file. If the position value is greater than the position of the last byte of the file, the position will point to the end of file.

This command can only be used on files open for byte read.

21.8 Byte.read.buffer

Syntax: Byte.read.buffer <file_table_nexp>, <count_nexp>, <buffer_svar> {, <charset_sexp>}

Reads the specified number of bytes (<count_nexp>) into the buffer string variable (<buffer_svar>) from the file. The string length (len(<buffer_svar>)) will be the number of bytes actually read. If the end of file is reached, the string length may be less than the requested count.

A buffer string is a special use of the BASIC! string. Each character of a string is 16 bits. When used as a buffer, one byte of data is written into the lower 8 bits of each 16-bit character. The upper 8 bits are 0. Extract the binary data from the string, one byte at a time, with the **ASCII()** or **UCODE()** functions.

The format of the buffer string read by this command is compatible with the **DECODE$()** function. If you know that part of your data contains an encoded string, you can extract the substring (using a function like **MID$()**), then pass the substring to **DECODE$()** to convert it to a BASIC! string.

Using the optional <charset_sexp>, you can choose between following character sets:

"_US-ASCII" and "_ISO-8859-1"(default).

In most cases the command **GrabFile** is a better choice.

21.9 Byte.read.byte

Syntax: Byte.read.byte <file_table_nexp> {,<nvar>}...

Reads bytes from a file. The <file_table_nexp> parameter is a file table number returned by a previous **Byte.open**. If the file number is -1 then the command throws a run-time error.

Places the byte(s) into the <nvar> parameter(s) as positive values, 0 <= byte <= 255. After the last byte in the file has been read, further attempts to read from the file return the value -1. The result of the **Byte.eof** command will be false after reading real data and true after reading at end-of-file (EOF).

This example reads a file and prints each byte, and prints "-1" at the end to show that the entire file has been read.

```
Byte.open r, file_number, "testfile.jpg"
Do
 Byte.read.byte file_number, byte
 Print byte
Until byte < 0
Byte.close file_number
```

21.10 Byte.read.number

Syntax: Byte.read.number <file_table_nexp> {,<nvar>...}

Reads numbers from the file specified by the file number parameter <file_table_nexp> and places them into the numeric variables in the "{,<nvar>}..." parameter list. Each number is a group of 8 bytes.

If the file does not have enough data available for all of the variables, the value of one or more <nvar> will be set to -1. This is indistinguishable from -1 read as actual data, except that the result of the **Byte.eof** command will be false for real data and true for EOF.

Normally this command is used only to read values written with **Byte.write.number**. You must be sure the file is positioned at the first byte of the eight-byte representation of a number, or you will get unexpected results.

21.11 Byte.truncate

Syntax: Byte.truncate <file_table_nexp>,<length_nexp>

Truncates the file to <length_nexp> bytes and closes the file. Setting the truncate length <length_nexp> larger than the current length (current write position - 1) has no effect.

This command can only be used on files open for byte write or append.

21.12 Byte.write.buffer

Syntax: Byte.write.buffer <file_table_nexp>, <buffer_sexp> {, <charset_sexp>}

Writes the entire contents of the string expression to the file. The string is assumed to be a buffer string holding binary data, as described in **Byte.read.buffer**. The writer discards the upper 8 bits of each 16-bit character, writing one byte to the file for each character in the string.

The **Byte.read.buffer** command and the **Encode$()** function always create these "buffer strings". You can construct one by using, for example, the **Chr$()** function with values less than 256.

If you use only ASCII characters in a string, you can use this function to write the string to a file. The output is the same as if you had written it with **Text.writeln**, except that it will have no added newline.

Using the optional <charset_sexp>, you can choose between following character sets: "_US-ASCII", "_UTF-8", "_UTF-16", "_UTF-16BE", "_UTF-16LE" and "_ISO-8859-1"(default).

If you want to save for international text, use "_UTF-8".

This command expects only character codes up to 255 for "_US-ASCII" or for binary data "_ISO-8859-1" and writes only one byte per character.

If you want to convert an integer to a byte array use bit shifting like this:

```
Fn.def integer2Str$(val)
 b0$ = Chr$(Band(val,255))
 b1$ = Chr$(Band(Shift(val, 8), 255))
 b2$ = Chr$(Band(Shift(val, 16), 255))
 b3$ = Chr$(Band(Shift(val, 24), 255))
 Fn.rtn b0$ + b1$ + b2$ + b3$
Fn.end
```

Using "_UTF-8", "_UTF-16", "_UTF-16BE", "_UTF-16LE" with **Byte.write.buffer** writes byte arrays with up to 4 bytes for each character.

If you get in trouble with binary data using **Byte.write.buffer**, try **Byte.write.byte** with your string. It is slow but it interprets character codes above 255 differently.

21.13 Byte.write.byte

Syntax: Byte.write.byte <file_table_nexp> {{,<nexp>}...{,<sexp>}}

Writes bytes to the file specified by the file number parameter <file_table_nexp>. The bytes are written from an optional comma-separated list of numeric expressions, a single optional string expression, or both.

- Numeric parameters: the command writes the low-order 8 bits of the value of each expression as a single byte.
- String parameters: the command writes the entire string to the file. Each character of the string is written as a single byte. See **Byte.write.buffer** for more information.
- If you have both numeric and string parameters, you may have only one string, and it must be last.
- The command accepts a string for backward compatibility. **Byte.write.buffer** is preferred.

Examples:

```
Byte.open w, f1, "tmp.dat"      % create a file
Byte.write.byte f1, 10               % write one byte
Byte.write.byte f1, 11, 12, 13   % write three bytes
Byte.write.byte f1, "Hello!"     % write six bytes
Byte.write.byte f1, 1,2,3,"abc"  % write six bytes
Byte.write.byte f1, "one", "two" % syntax error: only one string allowed
```

Note: If the bytes are written from a string expression then the expression is evaluated twice. You should not put calls to user-defined functions in the expression.

21.14 Byte.write.number

Syntax: Byte.write.number <file_table_nexp> {,<nexp>}...

Writes the values of the numeric expressions <nexp> to the file specified by the file number parameter <file_table_nexp>. This command always writes 8 bytes for each expression in the parameter list.

22 File Text Commands

The text file I/O commands are to be exclusively used for text (.txt) files. Text files are made up of lines of characters that end in CR (Carriage Return) and/or NL (New Line). The text file input and output commands read and write entire lines.

The default path is "<pref base drive>/rfo-basic/data/".

Note: It is essential to close an output file before your program ends. This should also be done before the RUN command is executed.

22.1 Text.close

Syntax: Text.close <file_table_nexp>

The previously opened file represented by <file_table_nexp> will be closed.

22.2 Text.eof

Syntax: Text.eof <file_table_nexp>, <lvar>

Report an opened file's end-of-file status. If the file is at EOF, the <lvar> is set true (non-zero). If the file or directory is not at EOF, the <lvar> is set false (zero).

A file opened for write or append is always at EOF. A file opened for read is not at EOF until all of the data has been read and then one more read is attempted. That read will have returned the string "EOF". **Text.position.set** may also position the file at EOF.

22.3 Text.open

Syntax: Text.open {r|w|a}, <file_table_nvar>, <path_sexp>

The file specified by the path string expression <path_sexp> is opened. The default path is "<pref base drive>/rfo-basic/data/". The <path_sexp> string is appended to the default path.

The first parameter is a single character that sets the I/O mode for this file:

Parameter	Mode	Notes
r	read	File exists: Reads from the start of the file. File does not exist: Error (see below).
w	write	File exists: Writes from the start of the file. Writes over any existing data. File does not exist: Creates a new file. Writes from the start of the file.
a	append	File exists: Writing starts after the last line in the file. File does not exist: Creates a new file. Writes from the start of the file.

A file table number is placed into the numeric variable <file_table_nvar>. This value is for use in subsequent **Text.readln**, **Text.writeln**, **Text.eof**, **Text.position.***, or **Text.close** commands.

If a file being opened for read does not exist then the <file_table_nvar> will be set to -1. The BASIC! programmer can check for this and either create the file or report the error to the user. Information about the error is available from the **GetError$()** function.

Opening a file for append that does not exist creates an empty file. Finally, opening a file for write that already exists deletes the contents of the file; that is, it replaces the existing file with a new, empty one.

22.4 Text.position.get

Syntax: Text.position.get <file_table_nexp>, <position_nvar>

Get the position of the next line to be read or written to the file. The position of the first line in the file is 1. The position is incremented by one for each line read or written. The position information can be used for setting up random file data access.

If a file is opened for append, the position returned will be relative to the end of the file. The position returned for the first line to be written after a file is opened will be 1. You will have to add these new positions to the known position of the end of the file when building your random access table.

22.5 Text.position.mark

Syntax: Text.position.mark {{<file_table_nexp>}{, <marklimit_nexp>}}

Marks the current line of the file, and sets the mark limit to <marklimit_nexp> bytes.

Both parameters are optional. If the file table index <file_table_nexp> is omitted, the default file is the last file opened; you must ensure that the last file opened was a **Text** file opened for reading. If the mark limit <marklimit_exp> is omitted, the default value is the file's current mark limit.

Please read **Files and Paths → Mark and Mark Limit**.

22.6 Text.position.set

Syntax: Text.position.set <file_table_nexp>, <position_nexp>

Sets the position of the next line to read. A position value of 1 will read the first line in the file.

Text.position.set can only be used for files open for text reading.

If the position value is greater than the number of lines in the file, the file will be positioned at the end of file. The position returned for Text.position.get at the EOF will be number of lines plus one in the file.

If you have marked a position in the file, you cannot set a position before the mark. You will not be notified that the position is different from what you requested (see **Text.position.mark)**.

22.7 Text.readln

Syntax: Text.readln <file_table_nexp> {,<svar>}...

Read the lines from the specified, previously opened file and write them into the <svar> parameter(s). If <file_table_nexp> is -1, indicating **Text.open** failed, or if it is not a valid file table number, then the command throws a run-time error.

After the last line in the file has been read, further attempts to read from the file place the characters "EOF" into the <line_svar> parameter(s). This is indistinguishable from the string "EOF" read as actual data, except that the result of the **Text.eof** command will be false after reading real data and true after reading at end-of-file (EOF).

This example reads an entire file and prints each line.

```
Text.open r, file_number, "testfile.txt"
Do
  Text.readln file_number, line$
  Print line$
Until line$ = "EOF"
Text.close file_number
```

The file will not automatically be closed when the end-of-file is read. Subsequent reads from the file will continue to return "EOF".

When you are done reading a file, the **Text.close** command should be used to close the file.

22.8 Text.writeln

Syntax: Text.writeln <file_table_nexp>, <parms same as Print>

The parameters that follow the file table pointer are parsed and processed exactly the same as the **Print** command parameters. This command is essentially a **Print** to a file.

Text.writeln with no parameters writes a newline. If a parameter line ends with a semicolon, the data is not written to the file. It is stored in a temporary buffer until the next **Text.writeln** command that does not end in a semicolon. There is only one temporary buffer no matter how many files you have open. If you want to build partial print lines for more than one file at a time, do not use **Text.writeln** commands ending with semicolons. Instead use string variables to store the temporary results. After the last line has been written to the file, the **Text.close** command should be used to close the file.

23 File ZIP Commands

The ZIP file I/O commands work with compressed files. ZIP is an archive file format that stores multiple directories and files, using a method of lossless data compression to save file space.

Use **Zip.dir** to get an array containing the names of all of the directories and files in an archive. Use the file names with **Zip.read** to extract files from the archive. **Zip.read** can not extract a directory. Use **Zip.write** to put files in a new archive. You can overwrite an existing ZIP file, but you cannot replace or add entries.

23.1 Zip.close

Syntax: Zip.close <file_table_nexp>

Closes the previously opened ZIP file.

23.2 Zip.count

Syntax: Zip.count <path_sexp>, <nvar>

Returns the number of entries inside the ZIP file located at <path_sexp>. The path is relative to "<pref base drive>/rfo-basic/data/".

The count is returned in the <nvar>. If the ZIP file does not exist, the returned count is 0.

23.3 Zip.dir

Syntax: Zip.dir <path_sexp>, Array$[] {,<dirmark_sexp>} {,<timeStamp_nexp>}

Returns the names of the files and directories inside the ZIP file located at <path_sexp>. The path is relative to "<pref base drive>/rfo-basic/data/".

The names are placed into Array$[]. The array is sorted alphabetically with the directories at the top of the list. If the array exists, it is overwritten, otherwise a new array is created. The result is always a one-dimensional array.

A directory is identified by a marker appended to its name. The default marker is the string "(d)". You can change the marker with the optional directory mark parameter <dirmark_sexp>. If you do not want directories to be marked, set <dirmark_sexp> to an empty string, "".

If the directory is empty, Zip.dir returns an array with one item and a string with one space (" ") in it.

Options of <timeStamp_nexp>:

- 0 no time stamp (default), sorted alphabetically
- 1 with time stamp as time in milliseconds + ":" + file name, but unsorted
- 2 with time stamp as time in milliseconds + ":" + file name, sorted in ascending order
- 3 with time stamp as time in milliseconds + ":" + file name, sorted in descending order

23.4 Zip.extract

Syntax: Zip.extract <destination_path_sexp>, <zipFile_path_sexp>{{, <skip_overwrite_nexp>}, ToExtractArray$[]}

Extracts the content of the Zip file specified by <zipFile_path_sexp> into the destination defined by <destination_path_sexp>. The destination directory will be created automatically. Older files and directories will be overwritten. The last modified date of the source files will also be extracted.

In the event of an error please note that the directories in the Zip file must be specified separately.

The optional <skip_overwrite_nexp> controls the overwriting of existing files. If set to 0, overwriting will always be done. If set to 1, overwriting will be skipped if the last modified date of the new file is older. If set to 2, no overwriting will take place. Default is 0.

The file paths described by ToExtractArray$[] will be extracted if the Zip file contains the right counterpart (**Zip.dir** <path_sexp>, Array$[], "").

Be sure, that the needed directories were already created.

Zip.extract has no access to **assets** and **res**ources in conjunction with APKs. If you want to read a ZIP that is in assets, you must first copy it from assets to an internal or external file system, possibly by **Byte.copy** or **File.move**. Then you can open this new file with **Zip.extract**.

Example:

```
File.mkDir "Suitcase"
Byte.open r, ftb, "cartman.png"
Byte.copy ftb, "Suitcase/cartman.png" % LastModified is current date and time.
Byte.close ftb

File.mkDir "Suitcase/insects"
Byte.open r, ftb, "fly.gif"
Byte.copy ftb, "Suitcase/insects/fly.gif"
Byte.close ftb
File.mkDir "Suitcase/things"
path$ = "Suitcase"
File.dir path$, mDirArray$[], "", 0, 1, "_DF" % In case of Zip.files,
                                              % only "_D" or "_DF"
Array.length al, mDirArray$[]
Dim mEntries$[al]
Dim mDirArrayZip$[al]
For i = 1 To al
 mDirArrayZip$[i] = path$ + "/" + mDirArray$[i]
 mEntries$[i] = mDirArrayZip$[i] % Extracted as
                                 % Extracted/Suitcase/insects/fly.gif
 ! mEntries$[i] = mDirArray$[i] % Extracted as Extracted/insects/fly.gif
Next
Zip.files mDirArrayZip$[], mEntries$[], "my.zip"
Zip.extract "Extracted", "my.zip"
Zip.dir "my.zip", zipArray$[]
Debug.on
Debug.dump.array zipArray$[]
```

23.5 Zip.files

Syntax: Zip.files FilesArray$[], EntriesArray$[], <zipFile_path_sexp> {{ <no_compr_sexp>}, <compr_type_sexp>}

FilesArray$[] must contain paths (directory and file names) of files to be compressed into a Zip file specified by <zipFile_path_sexp>. EntriesArray$[] must contain the paths (directory and file names) to be used. This way, input files can be renamed as they are placed in the Zip file. The paths in EntriesArray$[] start at the root of the Zip file. If renaming is not needed, the both arrays must be identical.

The date and time of the source files will also be stored.

File compressing needs time. Be patient if you want to zip your complete Data directory.

Sometimes it makes no sense to compress a compressed file. If the content of <no_comp_sexp> contains strings like "jpg/gif/mp3 …", then files described in EntriesArray$[] with an ending like ". jpg", ".

gif", ".mp3" … will be not compressed. The delimiter is the slash "/", because the comma can be part of a file name.

If you need fast access and no compression, set <no_comp_sexp> to "*" to skip the compression of all files.

The system constants in the optional <comp_type_sexp> enables easy access to compression types:

- _SYNC_FLUSH
- _NO_FLUSH
- _FULL_FLUSH
- _HUFFMAN_ONLY
- _FILTERED
- _DEFLATED
- _DEFAULT_STRATEGY
- _DEFAULT_COMPRESSION
- _BEST_COMPRESSION
- _BEST_SPEED
- _NO_COMPRESSION

Default is "" which equals _DEFAULT_COMPRESSION.

Note that _SYNC_FLUSH, _NO_FLUSH and _FULL_FLUSH are available beginning with Android 4.4 (KitKat). In these cases, earlier versions use _DEFAULT_COMPRESSION by default.

23.6 Zip.open

This command is deprecated. Use **Zip.files** and **Zip.extract** if possible.

Syntax: Zip.open {r|w}, <file_table_nvar>, <path_sexp>

The ZIP file specified by the path string expression <path_sexp> is opened. The path is relative to "<pref base drive>/rfo-basic/data/".

The first parameter is a single character that sets the I/O mode for this file:

Parameter	Mode	Notes
r	read	File exists: Reads from the start of the file. File does not exist: Error (see below).
w	write	File exists: Writes from the start of the file. Writes over any existing data. File does not exist: Creates a new file. Writes from the start of the file.

Note: unlike **Text.open** and **Byte.open**, **Zip.open** does not support append mode.

A file table number is placed into the numeric variable <file_table_nvar>. This value is for use in subsequent **Zip.read**, **Zip.write**, or **Zip.close** commands.

If there was an error opening the ZIP file, <file_table_nvar> is set to -1 with details available from the **GetError$()** function.

Zip.open has no access to **assets** and **res**ources in conjunction with APKs. If you want to read a ZIP that is in assets, you must first copy it from assets to an internal or external file system, possibly by using **Byte.copy**. Then you can open this new file with **Zip.extract**.

23.7 Zip.read

Syntax: Zip.read <file_table_nexp> ,<buffer_svar>, <file_name_sexp>

Reads the content of the file <file_name_sexp> from inside a ZIP file and puts the result byte(s) inside the buffer string variable <buffer_svar>.

The <file_table_nexp> parameter is a file table number returned by a previous **Zip.open** command. If the file number is -1 then the command throws a run-time error.

If the file <file_name_sexp> is not found in the ZIP, <buffer_svar> is set to "EOF".

If the user tries to read the content of a zipped directory, instead of a zipped file, then the command throws a run-time error. To read the contents of a zipped directory, use **Zip.dir**.

23.8 Zip.write

Syntax: Zip.write <file_table_nexp> ,<buffer_sexp>, <file_name_sexp>

Writes the contents of the string expression <buffer_sexp> into a ZIP, as a file named <file_name_sexp>. The ZIP is in a file previously opened with **Zip.open**.

The string <buffer_sexp> is assumed to be a buffer string holding binary data, typically a string coming from reading a local file with **Byte.read.buffer**.

24 Filter Commands

24.1 Filter

Syntax: Filter(<nexp>)

Returns the filter result based on <nexp>. A filter has to be selected (Filter.set) beforehand. This function can also used by the command Array.math with the option _(op)filter.

Returns NaN if something went wrong.

24.2 Filter.circular.convolution.imag

Syntax: Filter.circular.convolution.imag xReal[], xImag[], yReal, yImag[], outReal[], outImag[]

Computes the circular convolution of the given complex vectors. Each vector's length must be the same.

See also https://en.wikipedia.org/wiki/Circular_convolution

24.3 Filter.circular.convolution.real

Syntax: Filter.circular.convolution.real xVec[], yVec[], outVec[]

Computes the circular convolution of the given real vectors. Each vector's length must be the same.

See also https://en.wikipedia.org/wiki/Circular_convolution

24.4 Filter.fft

Syntax: Filter.fft Real[], Imag[]

Computes the discrete Fourier transform (DFT) of the given complex vector, storing the result back into the vector. The vector can have any length.

If the vector length has the power of 2, the Radix 2 algorithm is used, else the more complicated Bluestein algorithm runs for any length.

A vector is a one-dimensional array.

A complex vector has two components, a real part Real[] and an imaginary part Imag[].

If you need the start vector again, copy it before using this command.

24.5 Filter.ifft

Syntax: Filter.ifft Real[], Imag[]

Computes the **inverse** discrete Fourier transform (IDFT) of the given complex vector, storing the result back into the vector. This transform does not perform scaling, so the inverse is not a true inverse. The vector can have any length.

If the vector length has the power of 2, the Radix 2 algorithm is used, else the more complicated Bluestein algorithm runs for any length.

A vector is a one-dimensional array.

A complex vector has two components, a real part Real[] and an imaginary part Imag[].

If you need the start vector again, copy it before using this command.

24.6 Filter.set

Syntax: Filter.set <filter_bundle>

Selects the filter to be used next. The filter will be defined by the bundle <filter_bundle>. Initial values will be set to the default or predefined values.

Table of Bundle Keys		
Key	**Value**	**Description**
_Filter	_Average _Median _Bessel _Butterworth _Chebyshev0_5dB _Chebyshev1_0db _Chebyshev2_0dB _Chebyshev3_0db _SinglePol1st _SinglePol2nd (String)	Set the filter type. Always needed!
_Average Filter Returns the average of given First-In-First-Out (FIFO) Array values.		
_Length	(numeric)	Limits the length of the FIFO array. The first-in values are not removed until the array length is greater than defined by _Length. Default is 5.
_Section	Array (numeric)	Sets a default initial array section. The length of FIFO Array is also given by length of the initial array section. The length will be only overwritten, if _Length is greater than the initial array section length.
_Median Filter Returns the median of given First-In-First-Out Array values. See the command Array.median for more details.		
_Length	(numeric)	Limits the length of the FIFO array. The first-in values are not removed until the array length is greater than defined by _Length. Default is 5.
_Section	Array (numeric)	Sets a default initial array section. The length of FIFO Array is also given by length of the initial array section. The length will be only overwritten, if _Length is greater than the initial array section length.
For the next filters see also • https://www.electronics-tutorials.ws/filter/filter_1.html English • https://www.electronics-tutorials.ws/de/filtern/kapazitive-reaktanz.html German Note that waves can be manipulated also by mechanical devices like the combination of springs and dampers as an example. The exhaust tract of a car is nowadays usually also a sound filter. The outputs have an attenuation of 3 dB.		
_Bessel Filter		

Table of Bundle Keys		
Key	**Value**	**Description**
A filter often used for sensors.		
_Butterworth Filter A modern practical application of the filter is common in computer animation; it serves to reduce curve points without changing the general shape of the curve. (Source: Wikipedia)		
_Chebyshev0.5dB (also written as Tschebyscheff) Filter The characteristic of the Butterworth filter is that the transfer characteristics in the stopband and passband are smooth. In the case of the Chebyshev filter, a ripple in the transmission characteristics is allowed in order to increase the steepness of the filter. (Source: Hochschule Karlsruhe)		
_Chebyshev1.0dB Filter		
_Chebyshev2.0dB Filter		
_Chebyshev3.0dB Filter		
_FirstOrder or One-Pole Filter Like a combination of a resistor and a capacitor or a damper and a spring.		
_SecondOrder or Two-Pole Filter A combination of more different parts and may also an amplifier. Filters with higher orders are a combination of First Order and Second Order Filters.		
Advanced options for the filters from _Bessel until _SinglePol2nd		
_InitialValue	(numeric)	Sets an initial value. Default is 0.
_CutOffRate	(numeric)	Sets the cut off rate Nc. Tc: cut-Off time intervall Ts: sample time interval fc: cut-off frequency fs: sample frequency Nc = Tc / Ts = fs / fc Default is 1.
_PassType	_Low or _High (String)	Sets the filter types lowpass filter or highpass filter. Default is _Low.
Keep care for the next options, that only one option is active. The setting and effectiveness of logical filters is always a matter of trial and error and of course depends largely on the composition of the signal and the structure of the outliers.		
_AbsoluteError (Option)	0 or 1 (numeric)	A removal for absolute value outliers. For example, imagine a water tank with a water level sensor. The pump can only deliver a limited flow rate. The fill level in the measuring cycle can only move up by 3 cm (_UpperAbsolute). When fully open, the drain can lower the level by a maximum of 5 cm (_LowerAbsolute) per measurement cycle.

Table of Bundle Keys		
Key	**Value**	**Description**
		If the absolutely possible change in value is determined in one of these cases, the last calculated output value is summed with the respective maximum possible absolute change in value for further calculation. To return a useful result the following absolute error values have to be set. Default is 0.
_UpperAbsolute	(numeric)	The upper value _UpperAbsolute for _AbsoluteError. Default is 0.
_LowerAbsolute	(numeric)	The lower value _LowerAbsolute for _AbsoluteError. Default is 0.
_Limits (Option)	0 or 1 (numeric)	A removal for limit outliers. This means e.g. in the case of a water level report from a rain barrel, the level cannot be lower than the bottom of the barrel (_LowerLimit) and not higher than the upper edge of the barrel (_UpperLimit). If these limits are exceeded, the last calculated output value is used for further calculation. To return a useful result the following limits have to be set. Default is 0.
_UpperLimit	(numeric)	Sets he upper value _UpperLimit for _Limits. Default is 0.
_LowerLimit	(numeric)	Sets he lower value _LowerLimit for _Limits. Default is 0.
_Damper	> 0 (numeric)	The difference between the input value and the last output value is only accepted as 1/5 in the case of _Damper = 5, which additionally dampens a sudden increase in the output value in the event of outliers. In case of _Damper = 1 no damper is used. If _Damper < 1 and > 0, the opposite is true, that means the influence is amplified. Which is usually not desirable. Default is 1.
_Descending (Option)	0 or 1 (numeric)	This is a so-called "drag pointer filter". The output rises to the input value without delay and then slowly falls (descending) again with a delay. For example, you can use this in the case of an awning to react to wind events. If there are gusts, you have to react immediately (retract the awning), but the awning is not extended again with a time delay when the wind drops. Default is 0.
		$y_t^* = \alpha \cdot (y_t - y_{t-1}^*) + y_{t-1}^*$

25 Font Commands

Your program can use fonts loaded from files. At present, the only way to use a loaded font is with the **Gr.Text.SetFont** command.

25.1 Font.clear

Syntax: Font.clear

Clears the font list, deleting all previously loaded fonts. Any variable that points to a font becomes an invalid font pointer. An attempt to use any of the deleted fonts is an error.

Note: Font.delete leaves a marker in the font list, so pointers to other fonts will not be affected. That is why you can **Font.delete** the same font more than once. **Font.clear** clears the entire list, making all font pointer variables invalid. After executing **Font.clear**, you can't **Font.delete** any of the cleared fonts.

25.2 Font.delete

Syntax: Font.delete {<font_ptr_nexp>}

Deletes the previously loaded font specified by the font pointer parameter <font_ptr_nexp>. Any variable that points to the deleted font becomes an invalid font pointer. An attempt to use the deleted font is an error. It is not an error to delete the same font again.

If the font pointer is omitted, the command deletes the most-recently loaded font that has not already been deleted. Repeating this operation deletes loaded fonts in last-to-first order. It is not an error to do this when there are no fonts loaded.

25.3 Font.load

Syntax: Font.load <font_ptr_nvar>, <filename_sexp>

Loads a font from the file named by the <filename_sexp>. Returns a pointer to the font in the variable <font_ptr_nvar>. This pointer can be used to refer to the loaded font, for example, in a **Gr.Text.SetFont** command.

If the font file can not be loaded, the pointer will be set to -1. You can call the **GetError$()** function to get an error message.

26 FTP Client Commands

These FTP commands implement an FTP Client

26.1 Ftp.cd

Syntax: Ftp.cd <new_directory_sexp>{, <ok_svar>}

Changes the current working directory to the specified new directory.

The optional <ok_svar> returns an empty string if nothing fails. Otherwise an error message is returned.

26.2 Ftp.close

Syntax: Ftp.close {<ok_svar>}

Disconnects from the FTP server.

The optional <ok_svar> returns an empty string if nothing fails. Otherwise an error message is returned.

26.3 Ftp.delete

Syntax: Ftp.delete <filename_sexp>{, <ok_svar>}

Deletes the specified file.

The optional <ok_svar> returns an empty string if nothing fails. Otherwise an error message is returned.

26.4 Ftp.dir

Syntax: Ftp.dir <list_nvar> {{{,<dirmark_sexp>}, <timeStamp_nexp>}, <ok_svar>}

Creates a list of the names of the files and directories in the current working directory and places it in a BASIC! List data structure. A pointer to the new List is returned in the variable <list_nvar>.

A directory is identified by a marker appended to its name. The default marker is the string "(d)". You can change the marker with the optional directory mark parameter <dirmark_sexp>. If you do not want directories to be marked, set <dirmark_sexp> to an empty string, "".

Options for <timeStamp_nexp>:

- 0 no time stamp (default)
- 1 with time stamp as time in milliseconds + ":" + file name, but unsorted
- 2 with time stamp as time in milliseconds + ":" + file name, sorted in ascending order
- 3 with time stamp as time in milliseconds + ":" + file name, sorted in descending order

The optional <ok_svar> returns an empty string if nothing fails. Otherwise an error message is returned.

The following code can be used to print the file names in that list:

```
Ftp.dir file_list
List.size file_list,size

For i = 1 To size
 List.get file_list,i,name$
```

```
 Print name$
Next i
```

26.5 Ftp.get

Syntax: Ftp.get <source_sexp>, <destination_sexp>{, <ok_svar>}

The source file on the connected ftp server is downloaded to the specified destination file on the Android device.

You can specify a subdirectory in the server source file string.

The destination file path is relative to "`<pref base drive>/rfo-basic/data/`" If you want to download a BASIC! source file, the path would be, "`../source/xxx.bas`".

The optional <ok_svar> returns an empty string if nothing fails. Otherwise an error message is returned.

26.6 Ftp.mkDir

Syntax: Ftp.mkDir <directory_sexp>{, <ok_svar>}

Creates a new directory of the specified name.

The optional <ok_svar> returns an empty string if nothing fails. Otherwise an error message is returned.

26.7 Ftp.open

Syntax: Ftp.open <url_sexp>, <port_nexp>, <user_sexp>, <pw_sexp>{, <ok_svar>}

Connects to the specified url and port. Logs onto the server using the specified user name and password.

The optional <ok_svar> returns an empty string if nothing fails. Otherwise an error message is returned.

It is not possible to access an FTP server of hBasic or OliBasic.

Example:

```
ftp.open "ftp.dlptest.com", 21, "dlpuser", "rNrKYTX9g7z3RgJRmxWuGHbeu"
```

For more information, see https://dlptest.com/ftp-test/

Examples:

```
Ftp.open "ftp://ftp.dlptest.com", 21, "dlpuser", "rNrKYTX9g7z3RgJRmxWuGHbeu",
ok$
```

Or:

```
Ftp.open "ftps://ftp.dlptest.com", 21, "dlpuser", "rNrKYTX9g7z3RgJRmxWuGHbeu",
ok$
Ftp.open "ftpes://192.168.1.2", 2221, "demo", "demo"
```

In the last two cases, you open an FTP over Transport Layer Security (TLS) connection, so called FTPS. This implementation works in explicit mode (also known as FTPES).

Working as a client with implicit mode is not supported by Android.

For a FTP/FTPES/FTPS server on Android, the **WiFi FTP Server** is recommended.

```
(id=com.medhaapps.wififtpserver or id=com.medhaapps.wififtpserver.pro)
```

If you want start and stop a FTP server, you can also use the app **FTP-Server**.

```
(id=com.theolivetree.ftpserver or id=com.theolivetree.ftpserverpro)
```

Start and Stop by the following intents:

```
com.theolivetree.ftpserver.StartFtpServer
or
com.theolivetree.ftpserver.StartFtpServerPro

com.theolivetree.ftpserver.StopFtpServer
or
com.theolivetree.ftpserver.StopFtpServerPro
```

If you want to use SFTP use the App **andFTP** (id=lysesoft.andftp and id=lysesoft.andftppro[Key only]). This App can be controlled by intents.

More information at http://www.lysesoft.com/products/andftp/index.html.

For an SFTP server on Android, the **Ssh Server** is recommended.

```
(id=com.theolivetree.sshserver or id=com.theolivetree.sshserverpro)
```

Start and Stop by the following intents:

```
com.theolivetree.sshserver.StartSshServer
or
com.theolivetree.sshserverpro.StartSshServer
com.theolivetree.sshserver.StopSshServer
or
com.theolivetree.sshserverpro.StopSshServer
```

26.8 Ftp.put

Syntax: Ftp.put <source_sexp>, <destination_sexp>{, <ok_svar>}

Uploads specified source file to the specified destination file on connected ftp server.

The source file is relative to the directory, "`<pref base drive>/rfo-basic/data/`" If you want to upload a BASIC! source file, the file name string would be: "`../source/xxxx.bas`".

The destination file is relative to the current working directory on the server. If you want to upload to a subdirectory of the current working directory, specify the path to that directory. For example, if there is a subdirectory named "etc" then the filename, "/etc/name" would upload the file into that subdirectory.

The optional <ok_svar> returns an empty string if nothing fails. Otherwise an error message is returned.

26.9 Ftp.rename

Syntax: Ftp.rename <old_filename_sexp>, <new_filename_sexp>{, <ok_svar>}

Renames the specified old filename to the specified new file name.

The optional <ok_svar> returns an empty string if nothing fails. Otherwise an error message is returned.

26.10 Ftp.rmDir

Syntax: Ftp.rmDir <directory_sexp>{, <ok_svar>}

Removes (deletes) the specified directory if and only if that directory is empty.

The optional <ok_svar> returns an empty string if nothing fails. Otherwise an error message is returned.

27 FTP Server Commands

These commands control the built-in FTP server.

This server does not support an FTP over Transport Layer Security (TLS) connection, also called FTPS. Because of this, these commands differ in name from Hbasic commands.

Ftp.server.set Configures the server. (must restart server after)

Ftp.server.start Starts the server.

Ftp.server.stop Stops the server.

All commands return an exit code (not optional) which will be one of the following:

Return Codes	
0	Success
1	Could not start server
2	Server already running
3	Could not stop server
4	Server already closed
5	Bad port number (port number must be 1025 to 65535)
6	No internet permission

27.1 Ftp.server.set

Syntax: Ftp.server.set <return_code_nvar>{{{{, <port_nexp>}, <username_sexp>}, <password_sexp>}, <welcome_string_sexp>}

Configures the server. The return code will be delivered in <return_code_nvar>. To set the servers control port between 1025 and 65535 use <port_nexp>. The default port is 2121. The login user name from client to server is specified by the optional <username_sexp>. Default name is "olibasic". The login password from client to server is specified by the optional <password_sexp>. Default password is "olibasic".

Note that some FTP clients do not allow empty usernames and/or passwords.

Use lowercase for all parameters.

The optional welcome string from server to client can be set by <welcome_string_sexp>. Do not put any newlines ("\n") inside the welcome string.

The default message is: "Welcome to the OliBasic's FTP-Server".

The configuration does not take place until a new server starts. So do this before starting the server.

Example:

```
Ftp.server.set rc, "john", "mypass"
Ftp.server.set rc, "john", p$, "Welcome to myApp's Server"
Ftp.server.set ,,newpassword$
```

27.2 Ftp.server.start

Syntax: Ftp.server.start <return_code_nvar>{, <ip_svar>}

```
Starts the server.  The return code will be delivered in <return_code_nvar>.
The optional <ip_svar> returns the server's IP address plus the port number as
"d.d.d.d:port", like 192.168.1.105:2121.
```

Example:

```
Ftp.server.start rc, IP$
If rc > 0 Then Print "Error" Else Print "Server started on IP$"
```

27.3 Ftp.server.stop

Syntax: Ftp.server.stop <return_code_nvar>

Stops the server. The return code will be delivered in <return_code_nvar>.

Example:

```
Ftp.server.stop rc
If rc <> 0 Then Print "Error" Else Print "Server stopped"
```

28 GPS

These commands provide access to the raw location data reported by an Android device's GPS hardware. Before attempting to use these commands, make sure that you have GPS turned on in the Android Settings Application.

The Sample Program file, f15_gps.bas is a running example of the use of the GPS commands.

There are two kinds of data reports: GPS status and location data. They are not reported at the same time, so there is no guarantee that overlapping information matches. For example, the location data report includes a count of the satellites used in the most recent location fix. The same information can be derived from the GPS status report. If number of detected satellites changes between reports, the two numbers do not agree.

Older devices before than Android N (7) will still use the deprecated GPS API. For newer devices of Android N or greater, the library calls will use the GNSS API (which also includes GPS) for greater coverage and accuracy.

GNSS Constellations:

Type	Name	Nation
1	GPS	US
2	SBAS	Global
3	GLONASS	Russia
4	QZSS	Japan
5	BEIDOU	China
6	GALILEO	Europe
7	IRNSS	India

About Satellite Bundles and inview.

Satellite bundles are used also in the sample program `f15_gps.bas`.

You can get a count of satellites in view with **Gps.status**. You can also get a list of satellites with **Gps.status** or **Gps.satellites**. This is a list of bundle pointers with keys *prn, elevation*, *azimuth*, *snr* and *infix.*

If you provide a satellite list that was used before, then any satellites in this list are not removed, they are either updated (if they are in view), or (scrubbed and) moved to the bottom of the list. Therefore a reused list can get bigger and bigger, depending on how many unique satellites were picked up in the course of your run. So the list size may or may not be the same as the last **Gps.status** inview count.

About infix

This count can be obtained with **Gps.status**, **Gps.location**, or **Gps.satellites**. These infix counts are obtained from different libraries, so may not be in sync with each other.

An infix count is the number of satellites that were in view and used for a fix. This does not necessarily mean that you get a fix. It usually takes 4 or more satellites before you get a fix for data. Satellites used for a fix will have their bundle key *infix* set to 1.

PRN Number

Satellite bundles returned in the list by **Gps.status** and **Gps.satellites** still have a "prn" key, but its value is replaced with an OliBasic unique number. This is in order to accommodate multiple GNSS constellations, with the first digit identifying the constellation type. For example, 3023 identifies a

satellite from the GLONASS constellation.

The next 3 digits are described in this document:

https://developer.android.com/reference/android/location/GnssStatus#getSvid(int).

28.1 GPS Control Commands

28.1.1 Gps.close

Syntax: Gps.close

Disconnects from the GPS hardware and stops the location reports. GPS is automatically closed when you stop your BASIC! program. GPS is not turned off if you tap the HOME key while your GPS program is running.

28.1.2 Gps.open

Syntax: Gps.open {{<status_nvar>},{<time_nexp>},{<distance_nexp>}}, {<useNET_nexp>}, {<useLAST_lexp>}}

Connects to the GPS hardware and starts it reporting location information. This command must be issued before using any of the other GPS commands.

The three parameters are all optional; use commas to indicate missing parameters. The parameters are available for advanced usage. The most common way to use this command is simply **Gps.open**.

If you provide a status return variable <status_nvar>, it is set to 1.0 (TRUE) if the open succeeds, or 0.0 (FALSE) if the open fails. If the open fails, you may get information about the failure from the **GetError$()** function.

The time interval expression <time_nexp> sets the minimum time between location updates. The time is in milliseconds. If you do not set an interval, it defaults to the minimum value allowed by your Android device. This is typically one second. Note: to reduce battery usage, Android recommends a minimum interval of five minutes.

If you provide a distance parameter <distance_nexp>, it is a numerical expression that sets the minimum distance between location updates, in meters. That is, your program will not be informed of location changes until your device has moved at least as far as the minimum distance setting. If you do not set a distance, any location change that can be detected will be reported.

This command attempts to get an initial "last known location". If the GPS hardware does not report a last known location, BASIC! tries to get one from the network location service. If neither source can provide one, the location information is left empty. If you use GPS commands to get location information before the GPS hardware starts reporting current location information, you will get this "last known location" data. The last known location may be stale, hours or days old, and so may not be useful.

The option <useNET_nexp> sets whether the scans will use the network provider or not, where:

0 = Don't use network provider (i.e just use GPS/Gnss provider)
1 = Use both network provider and GPS/Gnss provider for each scan.
2 = Only use network provider (do not use GPS/Gnss provider).
other = default = 0 (same as legacy).

Network provider can still get you some data even if GPS/GNSS does not.

Note: The network provider still needs 'location' enabled by the user.

The option <useLAST_lexp> is a flag which remembers and uses the last-known-location (for the first scan only) of GPS.Open, where:

0 = OFF, do not use last known location (recommended)
1 = ON, use last known location for first scan.
other = default = 0 (unlike legacy)

If you do not get any readings, the last known location just sticks, until you get a valid reading, which can be confusing or worse, misleading, for your next **Gps.open**.

28.1.3 Gps.status

Syntax: Gps.status {{<status_var>}, {<infix_nvar>},{inview_nvar}, {<sat_list_nexp>}}

Returns the data from a GPS status report. The parameters are all optional; use commas to indicate omitted parameters.

This kind of report contains the type of the last GPS event and a list of the satellites that were detected by the GPS hardware when that event occurred. As a convenience, this command analyzes the satellite list to report how many satellites were detected ("in view") and how many of those were used in the last location fix ("in fix").

The GPS status report is not timestamped, and the first event reported to your program may be stale. Do not rely on the data from the first status report alone to determine when the GPS hardware gets a current location fix..

<status_var>: If provided, this variable returns the type of last GPS event that occurred. If you provide a numeric variable, the event type is reported as a number. If it is a string variable, the event type is reported as an English-language event name.

Event Number	Event Name	Meaning
1	Started	The GPS system has been started, no location fixed yet
2	Stopped	The GPS system has been stopped
3	First Fix	The GPS system has received its first location fix since starting
4	Updated	The GPS system has updated its location data

<infix_nvar>: If provided, this numeric variable returns the number of satellites used in the last location fix. This is the number of satellites in the satellite list describe below whose "infix" value is TRUE (non-zero). If the status report could not get a satellite list the number is unknown, so the variable is set to -1.

<inview_nvar>: If provided, this numeric variable returns the number of satellites detected by the GPS hardware. This is the number of satellites in the satellite list described below that have current data. It is not necessarily the size of the list. If the status report could not get a satellite list the number is unknown, so the variable is set to -1.

<sat_list_nexp>: If provided, the value of this numeric expression is used as a list pointer. If the value is not a valid numeric list pointer, and the numeric expression is a single numeric variable, then a new list is created and the numeric variable is set to point to the new list.

The satellite list is a list of bundle pointers. When the GPS system reports GPS status, it provides data collected from the satellites it can detect. The data from each satellite is put in a bundle. The satellite list has pointers to all of the satellite data bundles. You can use these pointers with any **Bundle**

command, just like any other bundle pointer.

If you provide an existing list, any bundles already in the list are cleared, except for the identifying pseudo-random number (PRN). Anything else in the list is discarded. Then the new satellite data is written into the satellite bundle. This is done so that a satellite that is lost and then regained will be remembered in the satellite bundle, but its stale data will not be kept.

The number of satellite bundles with complete data matches the value of the **<inview_var>**. These bundles are listed first. Any cleared bundles for satellites not currently visible are at the end of the list.

Each satellite bundle has five key/value pairs. All values are numeric. The value of "infix" is interpreted as logical (Boolean).

KEY	VALUE
prn	Pseudo-Random Number assigned to the satellite for identification
elevation	Elevation in degrees
azimuth	Azimuth in degrees
snr	Signal-to-noise ratio: a measure of signal strength
infix	TRUE (non-zero) if the satellite's data was used in the last location fix, else FALSE (0.0)

This is the only GPS command that returns information from both kinds of GPS data. The satellite count returned in <count_nvar> comes from the location data report, and the satellite list returned in the satellite list comes from the GPS status report. If nothing changes between reports, the number of satellites with infix set TRUE is the same as the satellite count value, but this condition cannot be guaranteed.

The satellite count value is also returned by the **Gps.location** command. The satellite list is also returned or updated by the **Gps.status** command. This command, **Gps.satellites**, is retained for backward-compatibility and for convenience.

For example, let's say the most recent GPS status report had data from three satellites with PRNs 4, 7, and 29.

```
Gps.open sts
Gps.status , , inView, sats
Debug.dump.list sats            % may print 7.0, 29.0, 4.0
```

Assume appropriate delays after the **Gps.open** and that **Debug** is enabled. Another GPS status report may report data from satellites 4, 7, and 8. Then the list dump might show 7.0, 4.0, 8.0, 29.0. The order is unpredictable, except that 29.0 will be last, because it is not currently visible. In both cases, the value of inView is 3.0.

Debug.dump.bundle of the satellite bundle with PRN 4 might show this:

```
Dumping Bundle 11
prn: 4.0
snr: 17.0
infix: 0.0
elevation: 25.0
azimuth: 312.0
```

28.2 GPS Location Commands

These commands report the values returned by the most recent GPS location report. The **Gps.satellites** command also returns the list of satellites contained in a GPS status report.

A location report contains:

- the location provider

- the number of satellites used to generate the data in the report
- the time when the data was reported
- an estimate of the accuracy of the location components
- the location components:
  - latitude
  - longitude
  - altitude
  - bearing
  - speed

There are individual commands available to read each element of a location report. If you use separate GPS commands to read different components of the location data, you don't know if the different components came from the same location report. To be certain of consistent data, get all of the location components from a single **Gps.location** command.

28.2.1 Gps.accuracy

Syntax: Gps.accuracy <nvar>

Returns the accuracy level in <nvar>. If non-zero, this is an estimate of the uncertainty in the reported location, measured in meters. A value of zero means the location is unknown.

28.2.2 Gps.altitude

Syntax: Gps.altitude <nvar>

Returns the altitude in meters in <nvar>.

28.2.3 Gps.bearing

Syntax: Gps.bearing <nvar>

Returns the bearing in compass degrees in <nvar>.

28.2.4 Gps.latitude

Syntax: Gps.latitude <nvar>

Returns the latitude in decimal degrees in <nvar>.

28.2.5 Gps.location

Syntax: Gps.location {{<time_nvar>}, {<prov_svar>}, {<count_nvar}, {<acc_nvar>}, {<lat_nvar>}, {<long_nvar>}, {<alt_nvar>}, {<bear_nvar>}, {<speed_nvar>}}

Returns the data from a single GPS location report. It returns all of the data provided by all of the individual GPS location component commands below, except that it does not return the satellite list of the **Gps.satellites** command.

The parameters are all optional; use commas to indicate missing parameters. All of the parameters are variable names, so if any parameter is not provided, the corresponding data is not returned.

The parameters are:

<time_nvar>: time of the location fix, in milliseconds since the epoch, as reported by **Gps.time**.
<prov_svar>: the location provider, as reported by **Gps.provider**.
<count_nvar>: the number of satellites used to generate the location fix, as reported by **Gps.satellites**.
<acc_nvar>: an estimate of the accuracy of the location fix, in meters, as reported by

Gps.accuracy.
<lat_nvar>: current latitude, in decimal degrees, as reported by **Gps.latitude**.
<long_nvar>: current longitude, in decimal degrees, as reported by **Gps.longitude**.
<alt_nvar>: current altitude, in meters, as reported by **Gps.altitude**.
<bear_nvar>: current bearing, in compass degrees, as reported by **Gps.bearing**.
<speed_nvar>: current speed, in meters per second, as reported by **Gps.speed**.

28.2.6 Gps.longitude

Syntax: Gps.longitude <nvar>

Returns the longitude in decimal degrees in <nvar>.

28.2.7 Gps.provider

Syntax: Gps.provider <svar>

Returns the name of the location provider in <svar>. Normally this is "**gps**". The first time you read location data, you get the last known location, which may come from either the GPS hardware or the network location service. If it came from the network, this command returns "**network**". If neither provider reported a last known location, the provider <svar> is the empty string, "".

28.2.8 Gps.satellites

Syntax: Gps.satellites {{<count_nvar>}, {<sat_list_nexp>}}

Returns the number of satellites used for the last GPS fix and a list of the satellites known to the GPS hardware.

Both parameters are optional. If you omit <count_nvar> but use <sat_list_nexp>, keep the comma.

If you provide a numeric variable <count_nvar>, it is set to the number of satellites used for the most recent location data. If the location report did not provide a satellite count, <count_nvar> is set to -1.

For a description of the satellite list pointer expression <sat_list_nexp>, see the <sat_list_nexp> parameter of the **Gps.status** command.

28.2.9 Gps.speed

Syntax: Gps.speed <nvar>

Returns the speed in meters per second in <nvar>.

28.2.10 Gps.time

Syntax: Gps.time <nvar>

Returns the time of the last GPS fix in milliseconds since "the epoch", January 1, 1970, UTC.

29 Graphics

29.1 Introduction

29.1.1 The Graphics Screen and Graphics Mode

Graphics are displayed on a new screen that is different from the BASIC! Text Output Screen. The Text Output Screen still exists and can still be written to. You can be returned to the text screen using the BACK key or by having the program execute the **Gr.front** command.

The **Gr.open** command opens the graphics screen and puts BASIC! into the graphics mode. BASIC! must be in graphics mode before any other graphics commands can be executed. Attempting to execute any graphics command when not in the graphics mode will result in a run-time error. The **Gr.close** command closes the graphics screen and turns off graphics mode. The graphics mode automatically turns off when the BACK key or MENU key is tapped. BASIC! will continue to run after the BACK key or MENU key is tapped when in graphics mode but the Output Console will be shown.

The BASIC! Output Console is hidden when the graphics screen is being displayed. No run-time error messages will be observable. A haptic feedback alert signals a run-time error. This haptic feedback will be a distinct, short buzz. Tap the BACK key to close the Graphics Screen upon feeling this alert. The error messages can then read from the BASIC! Output Console.

Use the **Gr.front** command to swap the front-most screen between the Output Console and the graphics screen.

Commands that use a new window or screen to interact with the user (**Input**, **Select** and others) may be used in graphics mode.

When your program ends, the graphics screen will be closed. If you want to keep the graphics screen showing, use a long pause or an infinite loop to keep the program from ending:

```
! Stay running to keep the graphics screen showing
Do
Until 0
```

Depending on your application, you may want to add a **Pause** to the loop to conserve battery power. Tap the BACK key to break out of the infinite loop. The BACK key ends your program unless you trap it with the **OnBackKey:** interrupt label.

29.1.2 Display Lists

Each command that draws a graphical object (line, circle, text, etc.) places that object on a list called the Object List. The command returns the object's Object Number. This Object Number, or Object Pointer, is the object's position in the Object List. This Object Number can be used to change the object on the fly. You can change the parameters of any object in the Object List with the **Gr.modify** command. This feature allows you easily to create animations without the overhead of having to recreate every object in the Object List.

To draw graphical objects on the graphics screen, BASIC! uses a Display List. The Display List contains pointers to graphical objects on the Object List. Each time a graphical object is added to the Object List, its Object Number is also added to the Display List. Objects are drawn on the screen in the order in which they appear in the Display List. The Display List objects are not visible on the screen until the **Gr.render** command is called.

You may use the **Gr.newDL** command to replace the current Display List with a custom display list array. This custom display list array may contain some or all of the Object Numbers in the Object List.

One use for custom display lists is to change the Z order of the objects. In other words you can use this feature to change which objects will be drawn on top of other objects.

See the Sample Program file, f24_newdl, for a working example of **Gr.newDL**.

29.1.3 Drawing Coordinates

The size and location of an object drawn on the screen are specified in pixels. The coordinates of the pixel at the upper-left corner of the screen are x = 0 (horizontal position) and y = 0 (vertical position). Coordinate values increase from left to right and from top to bottom of the screen.

Coordinates are measured with respect to the physical screen, not to anything on it. If you choose to show the Android Status Bar, anything you draw at the top of the screen is covered by the Status Bar.

29.1.4 Drawing into Bitmaps

You can draw into bitmaps in addition to drawing directly to the screen. You notify BASIC! that you want to start drawing into a bitmap instead of the screen with the **Gr.bitmap.drawInto.start** command. This puts BASIC! into the draw-into-bitmap mode. All draw commands issued while in this mode will draw directly into the bitmap. The objects drawn in this mode will not be placed into the Object List. The Object Number returned by a draw command while in this mode is invalid and should not be used for any purpose including **Gr.modify**.

The draw-into-bitmap mode is ended by the **Gr.bitmap.drawInto.end** command. Subsequent draw commands will place the objects on the Object List and object numbers in the Display List for rendering on the screen. If you wish to display the drawn-into bitmap on the screen, issue a **Gr.bitmap.draw** command for that bitmap. The drawn-into bitmap may be drawn at any time before, during or after the draw-into process.

29.1.5 Colors

BASIC! colors consist of a mixture of Red, Green, and Blue. Each component has a numerical value ranging from 0 to 255. Black occurs when all three values are zero. White occurs when all three values are 255. Solid Red occurs with a Red value of 255 while Blue and Green are zero.

Colors also have what is called an Alpha Channel. The Alpha Channel describes the level of opaqueness of the color. An Alpha value of 255 is totally opaque. No object of any color can show through an object with an Alpha value of 255. An Alpha value of zero renders the object invisible.

29.1.6 Paints

BASIC! holds drawing information such as color, font size, style and so forth, in Paint objects. The Paint objects are stored in a Paint List. The last created Paint object (the "Current Paint") is associated with a graphical object when a draw command is executed. The Paint tells the renderer (see **Gr.render**) how to draw the graphical object. The same Paint may be attached to many graphical objects so they will all be drawn with the same color, style, etc.

29.1.6.1 Basic usage

You can ignore Paints. This can keep your graphics programming simpler.

Each command that changes a drawing setting (**Gr.color**, **Gr.text.size**, etc.) affects everything you draw from that point on, until you execute another such command.

29.1.6.2 Advanced usage

You can control the Paint objects used to draw graphical objects. The extra complexity allows you to create special effects that are not otherwise possible. To use these effects, you must understand the Paint List and the Current Paint.

Each command that changes a drawing setting (**Gr.color**, **Gr.text.size**, etc.) creates or modifies a Paint. Each of these commands has an optional "Paint pointer" parameter. If you don't specify which Paint to use, the command first copies the Current Paint and then modifies it to make a new Paint, which then becomes the Current Paint. If you do specify which Paint to use, the command modifies only the specified Paint, leaving the Current Paint unchanged.

The Current Paint is always the last Paint on the Paint List. If you specify the Paint object, the pointer values -1 and 0 are special.

Paint pointer value -1 means the Current Paint. Using -1 is the same as omitting the Paint pointer parameter.

Paint pointer value 0 refers to a "working Paint". You can change it as often as you like without generating a new Paint. Paint 0 can not be attached to a graphical object. To make it useful, you must copy it (**Gr.paint.copy**) to another location in the Paint List, or to the Current Paint.

You can get a pointer to the current Paint with the **Gr.paint.get** command. You can get a pointer to the Paint associated with any graphical object by using the "paint" tag with **Gr.get.value** command. You can assign that Paint to any other graphical object by using the "paint" tag with **Gr.modify** command.

Paints can not be individually deleted. You may delete all Paint objects, along with all graphics objects, with **Gr.cls**.

The commands **Gr.paint.copy** and **Gr.paint.reset** operate directly on Paint objects.

29.1.7 Style

Most graphics objects are made up of two parts: an outline and the center. There is a subtle but important difference in the rules governing how each part is drawn. The two parts are controlled by the style setting of the Paint object, set by the style parameter of the **Gr.color** command.

Note: **Gr.point** and **Gr.line** are not affected by style. Only the STROKE part (outline) is drawn.

29.1.7.1 FILL

If you specify FILL (style 1), the center of the shape is filled as if it had an outline. That is, the center of the object is colored, but the pixels colored by STROKE may be left uncolored. The area that is colored depends on the coordinates of the shape. For example, consider a rectangle:

left: The left-most edge. All pixels with x-coordinate **left** are colored.
top: The upper-most edge. All pixels with y-coordinate **top** are colored.
right: One more than the right-most edge. All pixels with x-coord **right - 1** are colored.
bottom: One more than the lower-most edge. Pixels with y-coord **bottom - 1** are colored.

left and **top** values are "inclusive": pixels with x-coodinate **left** are colored. Pixels with y-coordinate **top** are colored.

right and **bottom** values are "exclusive": pixels with x-coodinate **right** are **not** colored. Pixels with y-coordinate **bottom** are **not** colored.

This is not what most people expect, but it is consistent with many operations in Java programming.

29.1.7.2 STROKE

If you specify STROKE (style 0), only the outline is drawn. The center of the shape is not colored. The width of the outline is controlled by the stroke weight setting of the Paint, set by the **Gr.stroke** command. If the stroke weight is 0 or 1 (they behave the same way), the outline is drawn exactly on the coordinates you provide. For example, if the shape is a rectangle:

left: The left-most edge. All pixels with x-coordinate **left** are colored.

top: The upper-most edge. All pixels with y-coordinate **top** are colored.
right: The right-most edge. All pixels with x-coord **right** are colored.
bottom: The lower-most edge. All pixels with y-coord **bottom** are colored.

This is probably what you expect.

When you increase the stroke weight, the lines get wider, but the centers of the lines do not change. Additional lines of pixels are colored on both sides of the outline. Increasing the stroke weight generally increases the area of the shape, making it larger than the dimensions you specify.

This may not be what you expect. In some drawing systems, increasing the line width grows the line inward only, toward the center of the shape, so the shape does not get any bigger.

29.1.7.3 STROKE and FILL

If you specify STROKE_AND_FILL (style 2), both parts of the shape are drawn and superimposed. That is, the outline is drawn as described in STROKE, and the object is filled in as described in FILL. Because both parts are the same color, and there are never uncolored pixels between the STROKE lines and the FILL area, the effect is to draw a single solid shape. Note that increasing the stroke weight generally makes the shape bigger than the dimensions you specify.

29.1.8 Hardware Accelerated Graphics

Current Android devices support hardware acceleration of some graphical operations. An app that can use the hardware Graphics Processor (GPU) may run significantly faster than one that cannot use the GPU.

If one of BASIC!'s graphical operations do not work with hardware acceleration, you can disable it with the **GR.set.acceleration** command. Hardware accelerated graphics is set to on by default.

See also **Gr.render**, **GR.set.acceleration** (on by default).

29.2 Graphics Setup Commands

29.2.1 Gr.brightness

Syntax: Gr.brightness <nexp>

Sets the brightness of the graphics screen. The value of the numeric expression should be between 0.01 (darkest) and 1.00 (brightest).

29.2.2 Gr.close

Syntax: Gr.close

Closes the opened graphics mode. The program will continue to run. The graphics screen will be removed. The text output screen will be displayed.

29.2.3 Gr.cls

Syntax: Gr.cls {<clear_all_nexp>}

Clears the graphics screen. Deletes all previously drawn objects; all existing object references are invalid. Deletes all existing Paints and resets all **Gr.color** or **Gr.text** {size|align|bold|strike|underline|skew} settings. Disposes of the current Object List and Display List and creates a new Initial Display

List.

Note: bitmaps are not deleted. They will not be drawn because no graphical objects point to them, but the bitmaps still exist. Variables that point to them remain valid.

The **Gr.render** command must be called to make the cleared screen visible to the user.

Normally, bitmaps and drawables are not deleted. They will not be drawn because no graphical objects point to them, but the bitmaps or drawables still exist, and variables that point to them remain valid.

However, if the optional <clear_all_nexp> is > 0, then all bitmaps and drawables are deleted also. In this case, all variables that point to them will not be valid.

29.2.4 Gr.color

Syntax: Gr.color {{alpha}{, red}{, green}{, blue}{, style}{, paint}{, xFermode}}

Sets the color and style for drawing objects. There are two ways to use this command.

- *Basic usage*: ignore the optional <paint> parameter. The new color and style will be used for whatever graphical objects are subsequently drawn until the next **Gr.color** command is executed.
- *Advanced usage*: The "basic usage" of this command always creates a new Paint. If you prefer, you can use the <paint> parameter to specify an existing Paint. The **Gr.color** command sets the color and style of that Paint, changing the appearance of any graphical object to which it is attached. The current Paint is not changed. See "Paints *Advanced Usage*" and the example below.

All of the parameters are optional. If a color component or the style is omitted, that component is left unchanged. For example, **Gr.color ,,0** sets only green to 0, leaving alpha, red, blue, and style as they were. Use commas to indicate omitted parameters.

Each of the four color components (alpha, red, green, blue) is a numeric expression with a value from 0 through 255. If a value is outside of this range, only the last eight bits of the value are used; for example, 257 and 1025 are both the same as 1. You can also use color definitions such as `"_127, Green", "_Blue", "#ff008080"` etc. See color definitions in section 2 Color Table.

Other possibilities are:

```
"_{<alpha>,}HSV<hue[0...360]>{{,<saturation [0...1]>},<valueOfBrightness
[0...1]>}
```

The string `"_200,HSV240"` returns the color blue by a color wheel angle of 240 degrees with alpha of 200.

See also: https://en.wikipedia.org/wiki/HSL_and_HSV and https://en.wikipedia.org/wiki/Color_wheel.

The style parameter, is a numeric expression that determines the stroking and filling of objects. The effect of this parameter is explained in detail in the "Style" sections. The possible values for <style_nexp> are shown in this table:

Value	Meaning	Description
0	STROKE	Geometry and text drawn with this style will be stroked (outlined), respecting the stroke-related fields on the paint.
1	FILL	Geometry and text drawn with this style will be filled, ignoring all stroke-related settings in the paint.
2	STROKE_AND_FILL	Geometry and text drawn with this style will be filled and stroked at the same time, respecting the stroke-related fields on the paint.

If you specify a value other than -1, 0, 1, or 2, then the style is set to 2. If you specify a style of -1, the

style is left unchanged, just as if the style parameter were omitted. If you never set a style, the default value is 1, FILL.

You can change the stroke weight with commands such as **Gr.set.stroke** (see below) and the various text style commands.

Example:

```
Gr.OPEN

! basic usage
Gr.color ,0,0,255,2          % opaque blue, stroke and fill
Gr.rect r1, 50,50,100,100    % draw two squares
Gr.rect r2, 100,100,150,150
Gr.color 128,255,0,0         % half-transparent red
Gr.rect r3, 75,75,125,125    % draw an overlapping square
Gr.render
Pause 2000

! advanced usage
Gr.get.value r1, "paint", p % get index of first Paint
Gr.color 255,0,255,0,,p     % change that Paint's color to opaque green
Gr.render                   % both r1 and r2 change
Pause 2000
Gr.rect r4, 125,125,175,175 % use current Paint, unchanged
Gr.render                   % still draws half-transparent red
Pause 2000
Gr.close
```

You can also use color names.

Example:

```
Gr.color "_Green", 1
```

sets the color to green and the style to fill.

The argument <xFermode> sets the Porter-Duff Compositing and Blend Modes. If you want to use it directly, the background alpha set by **Gr.open** has to be set to 0.

For a mask, use **Gr.bitmap.drawInto.start** and **Gr.bitmap.drawInto.end**.

Possible <xFermode> values are:

Index	Mode	Formula
-1	NORMAL	Default
0	CLEAR	[Sa, Sc] Destination pixels covered by the source are cleared to 0.
1	SRC	[Da, Dc] The source pixels replace the destination pixels.
2	DST	[Sa + (1 - Sa)*Da, Rc = Sc + (1 - Sa)*Dc] The source pixels are discarded, leaving the destination intact.
3	SRC_OVER	[Sa + (1 - Sa)*Da, Rc = Dc + (1 – Da)*Sc] The source pixels are drawn over the destination pixels.
4	DST_OVER	[Sa * Da, Sc * Da] The source pixels are drawn behind the destination pixels.
5	SRC_IN	[Sa * Da, Sa * Dc]

		Keeps the source pixels that cover the destination pixels,discards the remaining source and destination pixels.
6	DST_IN	[Sa * (1 - Da), Sc * (1 – Da)] Keeps the destination pixels that cover source pixels, discards the remaining source and destination pixels.
7	SRC_OUT	[Da * (1 - Sa), Dc * (1 – Sa)] Keeps the source pixels that do not cover destination pixels. Discards source pixels that cover destination pixels. Discards all destination pixels.
8	DST_OUT	[Da, Sc * Da + (1 - Sa) * Dc] Keeps the destination pixels that are not covered by source pixels. Discards destination pixels that are covered by source pixels. Discards all source pixels.
9	SRC_ATOP	[Sa, Sa * Dc + Sc * (1 – Da)] Discards the source pixels that do not cover destination pixels. Draws remaining source pixels over destination pixels.
10	DST_ATOP	[Sa + Da - 2 * Sa * Da, Sc * (1 - Da) + (1 - Sa) * Dc] Discards the destination pixels that are not covered by source pixels. Draws remaining destination pixels over source pixels.
11	XOR	[Sa + Da – Sa*Da, Sc*(1 - Da) + Dc*(1 - Sa) + min(Sc, Dc)] Discards the source and destination pixels where source pixels cover destination pixels. Draws remaining source pixels.
12	DARKEN	[Sa + Da – Sa*Da, Sc*(1 - Da) + Dc*(1 - Sa) + max(Sc, Dc)] Retains the smallest component of the source and destination pixels.
13	LIGHTEN	[Sa * Da, Sc * Dc] Retains the largest component of the source and destination pixel.
14	MULTIPLY	[Sa + Da - Sa * Da, Sc + Dc - Sc * Dc] Multiplies the source and destination pixels.
15	SCREEN	Saturate(S + D) Adds the source and destination pixels, then subtracts the source pixels multiplied by the destination.
16	ADD	Adds the source pixels to the destination pixels and saturates the result.
17	OVERLAY	Multiplies or screens the source and destination depending on the destination color.

For more information see:

https://developer.android.com/reference/android/graphics/PorterDuff.Mode.html

Example:

```
Gr.open 0
Gr.color 150, 0, 155,0,1, , -1
Gr.rect destinationR, 20,80,400,600
Gr.color 255, 255, 0,0,1, , 0
Gr.circle sourceC, 300, 300, 200
Gr.render
```

29.2.5 Gr.front

Syntax: Gr.front flag

Determines whether the graphics screen or the Output Console will be the front-most screen. If flag = 0, the Output Console will be the front-most screen and seen by the user. If flag <> 0, the graphics screen will be the front-most screen and seen by the user.

One use for this command is to display the Output Console to the user while in graphics mode. Use **Gr.front 0** to show text output and **Gr.front 1** to switch back to the graphics screen.

Note: When the Output Console is in front of the graphics screen, you can still draw (but not render) onto the graphics screen. The **Gr.front 1** must be executed before any **Gr.render**.

Print commands will continue to print to the Output Console even while the graphic screen is in front.

When running Android before version 5, you might have trouble bringing the application from background to front. In that case, try a user defined function like **ReorderToFront()** as shown in the **App.SAR** examples. You can use this example for Graphic and Console mode, as it returns to the actual used mode automatically. But there is no guarantee in Android versions prior to 5, the applications returns automatically to top.

You should know, that the Console mode waits until the Graphic mode is finished. If you use **Console.front** or **Gr.front 0** in Graphic mode, BASIC! will hang. You should instead try **Dialog.message**, **Dialog.select** or **Select**. See also their enhanced command descriptions.

29.2.6 Gr.open

Syntax: Gr.open {{alpha}{, red}{, green}{, blue}{, <Decors_nexp>}{, <Orientation_nexp>}}
{,<Camera_nexp>}

Opens the Graphics Screen and puts BASIC! into Graphics Mode. The color values become the background color of the graphics screen. The default color is opaque white (255,255,255,255).

All parameters are optional; use commas to indicate omitted parameters.

Each of the four color components is a numeric expression with a value from 0 through 255. If a value is outside of this range, only the last eight bits of the value are used; for example, 257 and 1025 are the same as 1. If any color parameter is omitted, it is set to 255. Also allowed are color definitions such as "_127,Green", "_Blue", "#ff008080", etc. See the color definitions in Appendix chapter 2 Color Table.

Android 7 and Android 9 made some changes regarding screen decors like the status bar.

Starting with Android 7, the background of the status bar is the same as the graphic screen. If you turn the status bar on and the background is white you will not see text or icons. So you should darken the background within the bounds of the status bar. See also **Gr.screen**.

Starting with Android 9, display cutouts (also called notches) are supported.

The Status Bar text will be shown on the graphics screen if <Decors_nexp> is true (not zero). If the <Decors_nexp> is 0 or not present, the Status Bar text will not be shown.

Up to Android 6, the background of the status bar was darkened to Black by default. But keep in mind that the coordinate system still starts at the left, top display corner. Thus you have to move your graphic objects down if needed.

If <Decors_nexp> is 1X[0]X, the navigation bar will be in the Immersive Sticky Mode, thus the navigation bar is only visible if you stroke a touch from the outside into the screen.

If <Decors_nexp> is 1X[1]X, the navigation bar will be displayed in the normal mode.

If <Decors_nexp> is 1X[2]X, the navigation bar will be in the Translucent Mode, thus the navigation bar and the content behind are visible.

If <Decors_nexp> is 1[0]XX, the drawing layout will be shrunk by the insets of the cut out(s). In this case the coordinate system starts in left, top **shrunk layout** corner.

If <Decors_nexp> is 1[1]XX, the drawing layout is like the display layout. If a cut out or round display edges are within your graphic, it will not be displayed (because of no display area) but this area will still be saved in a screenshot. See also **Gr.screen**.

If <Decors_nexp> is **negative** (like `Gr.open "_Blue", -myLayouBundle, -1`), then **Gr.open** expects a layout bundle defined by items of the second following table.

For graphic-screen size-calculation and layout dimensions refer to **Gr.screen**.

Upon opening graphics mode, the orientation will be determined by the <Orientation_nexp> value. <Orientation_nexp> values are the same as values for the **Gr.orientation** command (see below). If the <Orientation_nexp> is not present, the default orientation is Landscape.

It is strongly recommended to set desired orientation in the **Gr.open** command since using subsequent **Gr.orientation** and **Gr.screen** commands can often lead to problems.

The <Camera_nexp> parameter sets an optional camera view behind the graphics screen as follows:

- 0 = No camera view in background (default)
- 1 = Camera view with the main rear camera
- 2 = Camera view with front camera
- If > number of cameras, the camera with the highest id is used.

Currently, only the main rear camera and the front camera are supported.

Keep in mind that Android's numbering starts with 0 while Basic! starts with 1.

At this time, only Android 6+ is supported.

If no CAMERA permission is granted, <Camera_nexp> falls back to 0.

Keep in mind that Orientation = -1 is not allowed if Camera > 0.

If the app goes into background mode after **Gr.open** with camera mode, you must use **Gr.close** after detecting **OnBackground:** and **Background()**.

See also: **Gr.screen**, **Gr.statusbar**

Example:

```
Fn.def OpenGraphicDisplay()
 sBr = 1101
 Screen.size size[], realsize[], density
 ori = 0
 If size[1] < size[2] Then ori = 1
 cam = 1
 Gr.open 0, 10, 0, 0, sBr, ori , cam

 Gr.camera.getparam p$
 p$ = Replace$(p$, ";",";\n")
 Print p$
 s$ = "effect=mono"
```

```
 Gr.camera.setparam s$,1

 fl = 3
 Gr.camera.flash fl

 Gr.statusbar sHeight, sE
 Gr.screen w, h
 Gr.color 255,250,0,0,2
 yFromTop = sHeight + 50
 Gr.rect rc1, 50, (sHeight*sE)+ 50, w-50, (sHeight*sE) + 150, 10
 Gr.render
 Fn.rtn 1
Fn.end

OpenGraphicDisplay()
mGr = 1
bg = 0
zoomFactor = 100
Gr.camera.zoom zoomFactor
zoomFactorMax = zoomFactor
zoomDirection = 1

Do
 Pause 100
Until 0

OnGrTouch:
 If zoomFactor > (zoomFactorMax-0.1) Then zoomDirection = -1
 zoomFactor = zoomFactor + 0.2 * zoomDirection
 If zoomFactor < 1 Then zoomDirection = 1 : zoomFactor = 1
 Gr.camera.zoom zoomFactor
 BigD.round zoomFactor$, Str$(zoomFactor), 1, "HU"
 sel = -400
 Dialog.message "Zoom Factor", zoomFactor$, sel
 Gr.OnGrTouch.resume

OnBackground:
 If Background()
  If mGr = 1 Then
   Gr.close
   mGr = 0
  EndIf
  bg = 1
 Else
  If bg = 1 Then
   OpenGraphicDisplay()
   mGr = 1
   bg = 0
  EndIf
 EndIf
Background.resume
```

Table of Decor Options				
Decor = [1] [ ] [ ] [ ]		0	1	2
[1]	Place holder		In any case	
[ ]	Use the place beside the cut out	No	Yes	
[ ]	Navigation bar	Immersive Sticky Mode (Android 7+. Translucent is the fall back.)	Yes	Translucent

[ ]	Status bar	No	Yes	
Decor = [ ]	Status bar	No	Yes	

The optional options bundle <-Decors_nexp> controls the layouts of the Action and Navigation bars. In other words, a negative <Decors_nexp> is interpreted as a bundle pointer.

Table of Layout Control Options		
Key	Value	Description
_ShowActionbar	0 or 1 (numeric)	If 1, show the Action bar if it is not currently showing. It is needed to show titles and to change the background color of the Statusbar. If 0 (default), hide the Actionbar if it is currently activated.
_Title	String	Set the action bar's title.
_Subtitle	String	Set the action bar's subtitle.
_TitleShow	0 or 1 (numeric)	If 1, show the Action bar if it is not currently showing. It will resize application content to fit the new space available. If 0 (default), hide the Actionbar if it is currently showing. It will resize application content to fit the new space available.
_TitleIcon	Icon file path	Add a large icon to the notification content view. See http://romannurik.github.io/AndroidAssetStudio/index.html
_TitleHomeEnabled	0 or 1 (numeric)	Set whether to include the application home accordance in the action bar. Home is presented as an activity icon. Must be 1 if you want to show the icon. Must be 0 if you want to hide the icon. The default setting is API dependent.
_TitleBackground	Background file path	
_TitleHtml	0 or 1 (numeric)	Returns displayable styled text from the provided HTML string. But not all tags are supported. Uses parts of TagSoup library to handle real HTML, including all of the brokenness found in the wild. <b> <big> <font size="..." color="..." face="..."> <h1>, <h2>, <h3>, <h4>, <h5>, <h6> <i> <small> <strike>? < A.7 <sub> <sup> <tt>?

Table of Layout Control Options		
		<u> Replace Space with , & with &amp, < with &lt, > with &gt, " with &quot if necessary. Usable for Title and Subtitle. Keep in mind that the Action bar height will not be expanded.
_ShowStatusbar	0, 1 or 2 (numeric)	If 1 (default), the Status bar will be displayed. If 2, the Status bar will be transparently displayed. Minimum Lollipop 5.0 (API 21) required. If 0, the Status bar will be hidden to the background. Minimum Nougat 7.0 (API 24) required. Will be switched to option 2 or 1 if the current API level is lower.
_StatusbarColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Minimum Lollipop 5.0 (API 21) required. Note, the Actionbar has to be activated by **_ShowActionbar.**
_StatusbarLight	0 or 1 (numeric)	If 0 (default), the Status bar background is dark. In this case the **bar content** will be **light**. If 1, the Status bar background is light. In this case the **bar content** will be **dark**. Minimum Lollipop 5.0 (API 21) required.
_ShowNavigationbar	0, 1 or 2 (numeric)	If 1 (default), the Navigation bar will be displayed. If 2, the Navigation bar will be transparently displayed. Minimum Lollipop 5.0 (API 21) required. If 0, the Navigation bar will be hidden to the background. Minimum Nougat 7.0 (API 24) required. Will be switched to option 2 or 1 if the current API level is lower.
_NavigationbarColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Minimum Lollipop 5.0 (API 21) required.
_NavigationbarLight	0 or 1 (numeric)	If 0 (default), the Navigation bar background is

Table of Layout Control Options		
		dark. In this case the **bar content** will be **light**. If 1, the Navigation bar background is light. In this case the **bar content** will be **dark**. Minimum Lollipop 5.0 (API 21) required.
_Menu	Menu Bundle Pointer	Creates menu entries. A successful selection will be returned as a human readable JSON string. See the example at **Console.title** for more details.
_ExtendBesideNotch	0 or 1 (numeric)	If 0 (default), the space beside the notch is not used for graphics. If 1, the space beside the notch is used also. Minimum Pie 9.0 (API 28) required.
_CameraViewBounds	left, top, right, bottom (comma delimited string)	Defines the bounds of a background camera view in pixels.

29.2.7 Gr.orientation

Syntax: Gr.orientation <nexp>

The value of the <nexp> sets the orientation of screen as follows:

-1 = Orientation depends upon the sensors.
0 = Orientation is forced to Landscape.
1 = Orientation is forced to Portrait.
2 = Orientation is forced to Reverse Landscape.
3 = Orientation is forced to Reverse Portrait.

You can monitor changes in orientation by reading the screen width and height using the **Gr.screen** or **Screen** commands. However, if **Gr.open** has been called with the Camera option, then the camera orientation is not changed by the **Gr.orientation** command.

29.2.8 Gr.render

Syntax: Gr.render

This command causes the system to schedule a redraw of all the objects that are listed in the current working Display List and are not marked as hidden. It is not necessary to have a **Pause** command after a **Gr.render**. The **Gr.render** command will not complete until the contents of the Display List have been fully displayed.

Gr.render always waits until the next screen refresh. Most Android devices refresh the screen 60 times per second; your device may be faster or slower. Therefore, if you execute two consecutive **Gr.render** commands, there will be a delay of 16.7 milliseconds (on most devices) between the two commands.

For smooth animation, try to avoid doing more than 16.7 ms of work between **Gr.render** commands, to achieve the maximum refresh rate. This is not a lot of time for a program, so you may have to settle for a lower frame rate. However, there is no benefit to trying to render more often than 16.7 ms.

If your program is running in the background (see **Background()** function and **Home** command), **Gr.render** will not execute. It will pause your program until your program returns to the foreground.

Keep in mind that under some circumstances (software rendering) a graphic object can be displayed before **Gr.render** is called. Hardware rendering is the default option and can only changed by **Gr.set.acceleration**.

If the program has to do a lot of work in the background before the result should be displayed, do not draw or change any objects until the background work is complete.

29.2.9 Gr.scale

Syntax: Gr.scale x_factor, y_factor {{, x_distance{, y_distance}}}

Scale all drawing commands by the numeric x and y scale factors. This command is provided to allow you to draw in a device-independent manner and then scale the drawing to the actual size of the screen that your program is running on.

Example:

```
! Set the device independent sizes
di_height = 480
di_width = 800

! Get the actual width and height
Gr.open                    % defaults: white, no status bar, landscape
Gr.screen actual_w, actual_h

! Calculate the scale factors
scale_width = actual_w /di_width
scale_height = actual_h /di_height

! Set the scale
Gr.scale scale_width, scale_height
```

Now, start drawing based on di_height and di_width. The drawings will be scaled to fit the device running the program.

Scale all drawing commands by the numeric x and y scale factors and optionally translates by the numeric x and y distance. This command is provided to allow you to draw in a device-independent manner and then scale and translate the drawing to the actual size of the screen that your program is running on.

The translation will be executed before scaling. If you need translation after scaling, multiply factor by distance.

Example:

```
! Set the device independent sizes
di_height = 480
di_width = 800

! Get the actual width and height
Gr.open        % defaults: white, no status bar, landscape
Gr.screen actual_w, actual_h

! Calculate the scale factors
scale_width = actual_w /di_width
scale_height = actual_h /di_height

! Set the scale
Gr.scale scale_width, scale_height
```

Now you can start drawing based on di_height and di_width. The drawing will then be scaled to fit the device running the program.

29.2.10 Gr.screen

Syntax: Gr.screen <width_nvar>, <height_nvar>{{{{{, <density_nvar> } <isRound_lvar> }, <layout[ ]>}, <insets[ ]>}, <bounds[ ]>}

Returns the screen's width and height, and optionally its density, in the numeric variables. The density, in dots per inch (dpi), is a standardized Android density value (usually 120, 160, or 240, or 480 dpi), and not necessarily the real physical density of the screen.

If a **Gr.orientation** command changes the orientation, the width and height values from a previous **Gr.screen** command are invalid.

Android's orientation-change animation takes time. You may need to wait for a second or so after **Gr.open** or **Gr.orientation** before executing **Gr.screen**, otherwise the width and height values may be set before the orientation change is complete.

Gr.screen returns a subset of the information returned by the newer **Screen** command.

If the display is round, <isRound_lvar> returns 1.

The <layout[ ]> array returns the bounds of the current display view. (Left, Top, Right, Bottom).

It is strongly recommended to use the <layout[ ]> array for graphic-screen size-calculation!

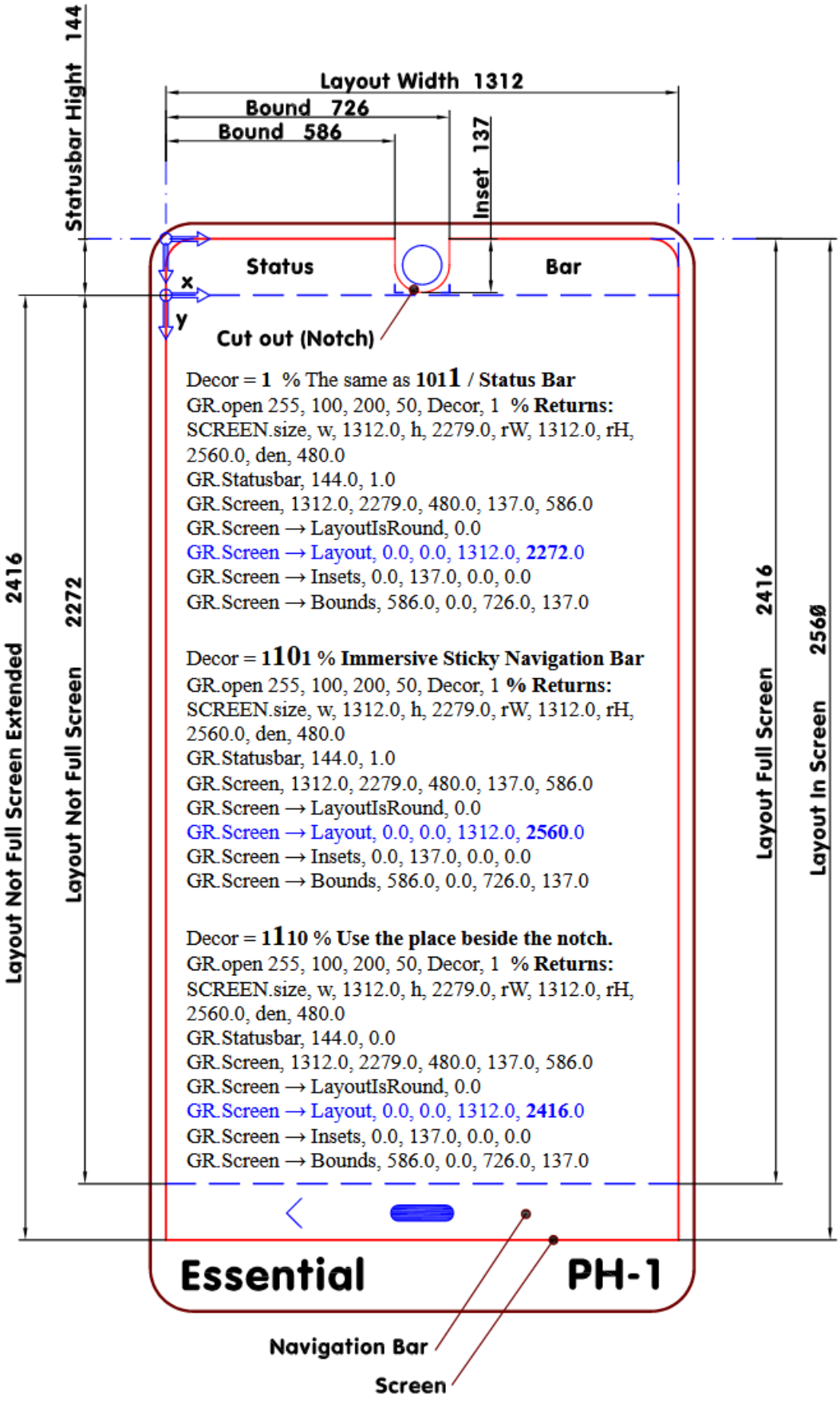
If the device has one or two notches and Android ≥ 9, then:

> The <insets[ ]> array returns the insets of the current display view ([Left, Top, Right, Bottom).
>
> The <bounds[ ]> array returns the bounds of the notch(es), ([Left, Top, Right, Bottom{,Left, Top, Right, Bottom}).

For example, if a notch with the dimensions 150 x 100 is in the upper left corner, then insets[] returns [150, 100, 0, 0] and bounds[] returns [0, 0, 150, 100].

The following illustration shows the various dimensions and parameters described above.

See also: **OnGrScreen:**, **Screen**


Statusbar Hight 144
Layout Width 1312
Bound 726
Bound 586
Inset 137
Status
Bar
x
y
Cut out (Notch)
Decor = 1 % The same as 1011 / Status Bar
GR.open 255, 100, 200, 50, Decor, 1 % Returns:
SCREEN.size, w, 1312.0, h, 2279.0, rW, 1312.0, rH, 2560.0, den, 480.0
GR.Statusbar, 144.0, 1.0
GR.Screen, 1312.0, 2279.0, 480.0, 137.0, 586.0
GR.Screen → LayoutIsRound, 0.0
GR.Screen → Layout, 0.0, 0.0, 1312.0, 2272.0
GR.Screen → Insets, 0.0, 137.0, 0.0, 0.0
GR.Screen → Bounds, 586.0, 0.0, 726.0, 137.0
Decor = 1101 % Immersive Sticky Navigation Bar
GR.open 255, 100, 200, 50, Decor, 1 % Returns:
SCREEN.size, w, 1312.0, h, 2279.0, rW, 1312.0, rH, 2560.0, den, 480.0
GR.Statusbar, 144.0, 1.0
GR.Screen, 1312.0, 2279.0, 480.0, 137.0, 586.0
GR.Screen → LayoutIsRound, 0.0
GR.Screen → Layout, 0.0, 0.0, 1312.0, 2560.0
GR.Screen → Insets, 0.0, 137.0, 0.0, 0.0
GR.Screen → Bounds, 586.0, 0.0, 726.0, 137.0
Decor = 1110 % Use the place beside the notch.
GR.open 255, 100, 200, 50, Decor, 1 % Returns:
SCREEN.size, w, 1312.0, h, 2279.0, rW, 1312.0, rH, 2560.0, den, 480.0
GR.Statusbar, 144.0, 0.0
GR.Screen, 1312.0, 2279.0, 480.0, 137.0, 586.0
GR.Screen → LayoutIsRound, 0.0
GR.Screen → Layout, 0.0, 0.0, 1312.0, 2416.0
GR.Screen → Insets, 0.0, 137.0, 0.0, 0.0
GR.Screen → Bounds, 586.0, 0.0, 726.0, 137.0
Layout Not Full Screen Extended 2416
Layout Not Full Screen 2272
Layout Full Screen 2416
Layout In Screen 2560
Essential
PH-1
Navigation Bar
Screen

29.2.11 Gr.set.acceleration

Syntax: Gr.set.acceleration <mode_nvar>

When there are no hardware acceleration settings in preferences and APK settings, this command overwrites your device's settings. Use this command with care.

It's good to place this command directly after the **Gr.open** command.

If <mode_nvar> = 0 (SOFTWARE): The graphic view is rendered in software into a bitmap.

If <mode_nvar> = 1 (HARDWARE): The view is rendered in hardware into a hardware texture if the application is hardware accelerated.

If <mode_nvar> = 2 (NONE): The view is rendered normally and is not backed by an off-screen buffer.

The default behavior is the mode from the APK-xml or set in the preferences.

The advantage of hardware acceleration is higher speed, and lower power consumption.

If you cannot use hardware acceleration, because some details are not displayed, proceed as follows. Create your graphic, if nothing happened in one or two seconds, take a screenshot and display the saved bitmap on top if necessary. Switch to hardware acceleration and wait for an event by **OnGrTouch:**. Now switch back to software rendering.

For more information see: https://developer.android.com/guide/topics/graphics/hardware-accel#java.

Gr.render is the same as the Java function **invalidate()**.

Look also under Unsupported Drawing Operations.

This command only works for Android 6+.

29.2.12 Gr.set.antialias

Syntax: Gr.set.antialias {{<lexp>}{,<paint_nexp>}}

Turns antialiasing on or off on objects drawn after this command is issued:

- If the value of the antialias setting parameter <lexp> is false (0), AntiAlias is turned off.
- If the parameter value is true (not zero), AntiAlias is turned on.
- If the parameter is omitted, the AntiAlias setting is toggled.

AntiAlias should generally be on. It is on by default.

AntiAlias must be off to draw single-pixel pixels and single-pixel-wide horizontal and vertical lines.

You may use the optional Paint pointer parameter <paint_nexp> to specify a Paint object to modify. Normally this parameter is omitted. See **Gr.color**, *Advanced usage* for more information.

29.2.13 Gr.set.cap

Syntax: Gr.set.cap {{<cap_nexp>}{,<paint_nexp>}}

Sets the line caps of objects drawn after this command is issued.

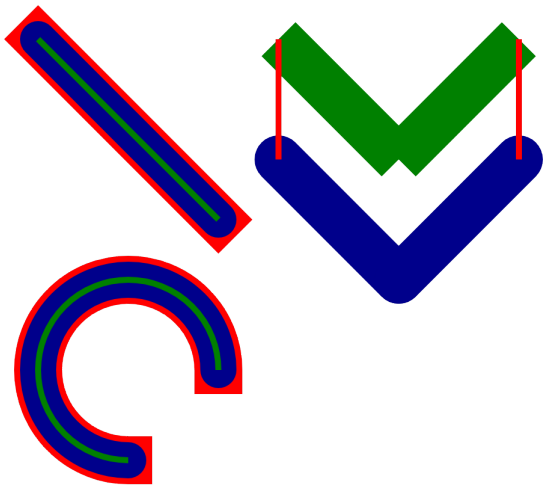
Possible values for <cap_nexp> are shown in this table:

Value	Meaning	Description
0	BUTT	The stroke ends with the path, and does not project beyond it.

1	ROUND	The stroke projects out as a semicircle, with the center at the end of the path.
2	SQUARE	The stroke projects out as a square, with the center at the end of the path.

Example:

```
Gr.open 80,0,0,0,0,1
Gr.arc Gr.set.cap 2
Gr.set.stroke 80
Gr.color 255,255,0,0
Gr.line nn,100,100,400,400
Gr.color 255,255,0,0,0 % Red with fill mode = 0, now the default one
Gr.arc cc,100,500,400,800, 0, -270, 0

Gr.set.stroke 60
Gr.set.cap 1
Gr.color "_DarkBlue"
Gr.line nn,100,100,400,400
Gr.arc cc,100,500,400,800, 0, -270, 0

Gr.set.stroke 10
Gr.color "_Green"
Gr.set.cap 0
Gr.line nn,100,100,400,400
Gr.arc cc,100,500,400,800, 0, -270, 0

Gr.set.stroke 80
Gr.line nn,500,100,700,300
Gr.line nn,700,300,900,100

Gr.set.cap 1
Gr.color "_DarkBlue"
Gr.line nn,500,300,700,500
Gr.line nn,700,500,900,300

Gr.set.stroke 10
Gr.set.cap 0
Gr.color "_Red"
Gr.line  nn,500,100,500,300
Gr.line  nn,900,100,900,300

Gr.render
```



29.2.14 Gr.set.stroke

Syntax: Gr.set.stroke {{<nexp>}{,<paint_nexp>}}

Sets the line width of objects drawn after this command is issued. The <nexp> value must be >=0. Zero produces the thinnest line and is the default stroke value.

The thinnest horizontal lines and vertical lines will be two pixels wide if AntiAlias is on. Turn AntiAlias off to draw single-pixel-wide horizontal and vertical lines.

Pixels drawn by the **Gr.set.pixels** command will be drawn as a 2x2 matrix if AntiAlias is on. To draw single-pixel pixels, set AntiAlias off and set the stroke = 0.

You may use the optional Paint pointer parameter <paint_nexp> to specify a Paint object to modify. Normally this parameter is omitted. See **Gr.color**, *Advanced usage* for more information.

29.2.15 Gr.statusbar

Syntax: Gr.statusbar {<height_nvar>} {, showing_lvar}

Returns information about the Status Bar. If the **height** variable <height_nvar> is present, it is set to

the nominal height of the Status Bar. If the **showing** flag <showing_lvar> is present, it is set to **0** (false, not showing) or **1** (true, showing) based on on how Graphics Mode was opened.

The parameters are both optional. If you omit the first parameter but use the second, you must keep the comma.

29.2.16 Gr.statusbar.show

Syntax: Gr.statusbar.show <nexp>

This command has been deprecated. To show the status bar on the graphics screen, use the optional fifth parameter in **Gr.open**.

29.2.17 Is_Gr

Syntax: Is_Gr()

Is_Gr() is a function (not a command) that returns the graphic mode status. If the graphic mode is enabled, it returns 1. But if the graphic mode is not open, it returns 0.

29.3 Graphics Object Creation Commands

These commands create graphical objects and add them to the Object List, also adding their Object Numbers to the Display List. You create each object with parameters that describe what to draw and where. Once it is created, you can read back its parameters by name with the **Gr.get.value** command. You can change any parameter with the **Gr.modify** command. The parameters you can modify are listed with each command's description. Along with the parameters listed with each command, every graphical object has two other modifiable parameters, "paint" and "alpha". See the **Gr.modify** and **Gr.paint.get** command descriptions for more details.

There are three commands that create graphical objects that are not in this section: **Gr.text.draw**, **Gr.bitmap.draw**, and **Gr.clip**.

29.3.1 Gr.arc

Syntax: Gr.arc <obj_nvar>, left, top, right, bottom, start_angle, sweep_angle, fill_mode

Creates an arc-shaped object. The arc will be created within the rectangle described by the parameters. It will start at the specified start_angle and sweep clockwise through the specified sweep_angle. The angle values are in degrees.

The effect of the fill_mode parameter depends on the **Gr.color** style parameter:

- Style 0, fill_mode false: Only the arc is drawn.
- Style 0, fill_mode true: The arc is drawn with lines connecting each endpoint to the center of the bounding rectangle. The resulting closed figure is not filled.
- Style non-0, fill_mode false: The endpoints of the arc are connected by a single straight line. The resulting figure is filled.
- Style non-0, fill_mode true: The arc is drawn with lines connecting each endpoint to the center of the bounding rectangle. The resulting closed figure is filled.

The <obj_nvar> returns the Object List object number for this arc. This object will not be visible until the **Gr.render** command is called.

The **Gr.modify** parameters for **Gr.arc** are: "left", "top", "right", "bottom", "start_angle", "sweep_angle" and "fill_mode". The value for "fill_mode" is either false (0) or true (not 0).

29.3.2 Gr.arcpoly

Syntax: Gr.arcpoly <obj_nvar>, <list_pointer_nexp> {{, x, y}, <closed_nexp>}

Creates an object that draws a closed polygon of any number of sides as arc-segments or lines. The **<obj_nvar>** returns the Object List object number for this polygon. This object will not be visible until the next **Gr.render**.

<list_pointer_nexp> is an expression that points to a list data structure. The list contains **x, y** coordinate pairs. The first coordinate pair defines the point at which the polygon drawing starts. Each subsequent pair of coordinates defines an arc or a line that is drawn from the previous pair of coordinates (1) via this pair of coordinates (2) to an end point defined by a further pair of coordinates (3).

A line from coordinates (1) to coordinates (3) is drawn if coordinates (1), coordinates (2) and coordinates (3) are co-linear, in this case also via (2);

coordinates (1) and coordinates (2) are equal;

coordinates (2) and coordinates (3) are equal or

coordinates (3) are missed, in this case to (2).

A final line drawn from the last point back to the first closes the polygon. If you do not want to close the polygon **<closed_nexp>** has to be 0. Default is <> 0. Note that the fill will still be drawn if a fill is specified in **Gr.color** under Style (1 or 2).

If the optional **x, y** expression pair is present, the values will be added to each of the x and y coordinates of the list. This provides the ability to move the polygon array around the screen. The default x, y pair is 0,0. Negative values for x and y are valid.

The polygon curve width, curve color, alpha and fill are determined by previous **Gr.color** and **Gr.set.stroke** commands just like any other drawn object. These attributes are owned by the **arcpoly** object, not by the list. If you use the same list in different **Gr.arcpoly** commands, the color, stroke, etc., may be different.

You can change the arc polygon (add, delete, or move points) by manipulating the list with **List** commands. You can change to a different or a changed list of points using **Gr.modify** with "list" as the tag parameter. Changes are not visible until the **Gr.render** command is called.

When you create an arc polygon with **Gr.arcpoly** or attach a new list with **Gr.modify**, the list must have an even number of values and at least two coordinate pairs (four values). These rules are enforced with run-time errors. The rules cannot be enforced when you modify the list with **List** commands. Instead, if you have an odd number of coordinates, the last is ignored. If you have only one point, **Gr.render** draws nothing.

The **Gr.modify** parameters are "x", "y", "list", "paint" and "alpha".

See also **Within()**, **Gr.target.modify**, **Gr.poly** and **Gr.path**.

29.3.3 Gr.circle

Syntax: Gr.circle <obj_nvar>, x, y, radius

Creates a circle object. The circle will be created with the given radius around the designated center (x,y) coordinates. The circle will or will not be filled depending upon the **Gr.color** style parameter. The <obj_nvar> returns the Object List object number for this circle. This object will not be visible until the **Gr.render** command is called.

The **Gr.modify** parameters for **Gr.circle** are "x", "y", and "radius".

29.3.4 Gr.line

Syntax: Gr.line <obj_nvar>, x1, y1, x2, y2

Creates a line object. The line will start at (x1,y1) and end at (x2,y2). The <obj_nvar> returns the Object List object number for this line. This object will not be visible until the **Gr.render** command is called.

The thinnest horizontal lines and vertical lines are drawn with **Gr.set.stroke 0**. These lines will be two pixels wide if AntiAlias is on. Turn AntiAlias off to draw single-pixel wide horizontal and vertical lines.

The **Gr.modify** parameters for **Gr.line** are: "x1", "y1", "x2" and "y2".

29.3.5 Gr.oval

Syntax: Gr.oval <obj_nvar>, left, top, right, bottom

Creates an oval-shaped object. The oval will be located within the bounds of the parameters. The oval will or will not be filled depending upon the **Gr.color** style parameter. The <obj_nvar> returns the Object List object number for this oval. This object will not be visible until the **Gr.render** command is called.

The **Gr.modify** parameters for **Gr.oval** are: "left", "top", "right" and "bottom".

29.3.6 Gr.point

Syntax: Gr.point <obj_nvar>, x, y

Creates a point object. The point will be located at (x,y). The <obj_nvar> returns the Object List object number for this point. This object will not be visible until the **Gr.render** command is called.

The appearance of the point object is affected by the current stroke weight and the AntiAlias setting. The object is rendered as a square, centered on (x,y) and as big as the current stroke. If AntiAlias is on, it will blur the point, making it larger and dimmer. To color a single pixel, use **Gr.set.stroke 0** and **Gr.set.antialias 0**.

The **Gr.modify** parameters for **Gr.point** are: "x" and "y".

29.3.7 Gr.poly

Syntax: Gr.poly <obj_nvar>, list_pointer {,x, y} {{{{, x, y}, <closed_nexp>}, <pointsPerPoly_nexp>}, <paintPointerList_nexp>}

Creates an object that draws a closed polygon of any number of sides. The <obj_nvar> returns the Object List object number for this polygon. This object will not be visible until the next **Gr.render**.

The list_pointer is an expression that points to a List data structure. The list contains x,y coordinate pairs. The first coordinate pair defines the point at which the polygon drawing starts. Each subsequent coordinate pair defines a line drawn from the previous coordinate pair to this coordinate pair. A final line drawn from the last point back to the first closes the polygon.

If the optional **x, y** expression pair is present, the values will be added to each of the x and y coordinates of the list. This provides the ability to move the polygon array around the screen. The default x, y pair is 0,0. Negative values for x and y are valid.

The polygon line width, line color, alpha and fill are determined by previous **Gr.color** and **Gr.set.stroke** commands just like any other drawn object. These attributes are owned by the **poly** object, not by the

list. If you use the same list in different **Gr.poly** commands, the color, stroke, etc., may be different.

You can change the polygon (add, delete, or move points) by directly manipulating the list with **List** commands. You can change to a different list of points using **Gr.modify** with "list" as the tag parameter. Changes are not visible until the **Gr.render** command is called.

When you create a polygon with **Gr.poly** or attach a new list with **Gr.modify**, the list must have an even number of values and at least two coordinate pairs (four values). These rules are enforced with run-time errors. The rules cannot be enforced when you modify the list with **List** commands. Instead, if you have an odd number of coordinates, the last is ignored. If you have only one point, **Gr.render** draws nothing.

The **Gr.modify** parameters are "x", "y" and "list".

See the Sample Program file, f30_poly, for working examples of **Gr.poly**.

If you do not want to close the polygon, set <closed_nexp> to 0. Default is <> 0. Note that the fill will still be drawn if a fill is specified in **Gr.color** under Style 1 or 2.

The **Gr.modify** parameters are "x", "y", "list", "paint" and "alpha".

For fast access the list <list_pointer_nexp> can be divided by <pointsPerPoly_nexp>. In this case more than one polygon can be created and the **Gr.paints** of the polygons can be optionally set by the list <paintPointerList_nexp>. If this list has less entries than created polygons, the list will be start again at the first entry. Note: if you need only one color, <paintPointerList_nexp> is not needed. If you use **Gr.modify** later, you also can set <paintPointerList_nexp>. But there is only one way back if <paintPointerList_nexp> has one or more entries with the same **Gr.paint**.

Note that changes in the List of <pointsPerPoly_nexp> will be mapped by each following **Gr.render** command **without** a **Gr.modify** before.

29.3.8 Gr.rect

Syntax: Gr.rect <ob_nvar>, left, top, right, bottom {, rx, ry}

Creates a rectangle object. The rectangle will be located within the bounds of the parameters. The rectangle will or will not be filled depending upon the **Gr.color** style parameter. The <obj_nvar> returns the Object List object number for this rectangle. This object will not be visible until the **Gr.render** command is called.

The **Gr.modify** parameters for **Gr.rect** are: "left", "top", "right" and "bottom".

The rx and ry parameters can be used to specify rounded corners. Use rx to round the corner in the x direction and ry in the y direction. So you can round the rectangle as an ellipse. If only rx is given, then ry will have the same value as rx.

The **Gr.modify** parameters for **Gr.rect** are: "left", "top", "right", "bottom", "rx" and "ry".

29.3.9 Gr.set.pixels

Syntax: Gr.set.pixels <obj_nvar>, pixels[{<start>,<length>}] {,x,y}

Inserts an array of pixel points into the Object List. The array (pixels[]) or array segment (pixels[start, length]) contains pairs of x and y coordinates for each pixel. The pixels[] array or array segment may be any size but must have an even number of elements.

If the optional x,y expression pair is present, the values will be added to each of the x and y coordinates of the array. This provides the ability to move the pixel array around the screen. The default values for

the x,y pair is 0,0. Negative values for the x,y pair are valid.

Pixels will be drawn as 2x2 matrix pixels if AntiAlias is on and the stroke = 0. To draw single-pixel pixels, set AntiAlias off and set the stroke = 0. AntiAlias in on by default.

The <obj_nvar> returns the Object List object number for the object. The pixels will not be visible until the **Gr.render** command is called.

The **Gr.modify** parameters for this command are "x" and "y".

In addition to modify, the individual elements of the pixel array can be changed on the fly. For example:

```
Pixels[3] = 120
Pixels[4] = 200
```

will cause the second pixel to be located at x = 120, y = 200 at the next rendering.

29.4 Graphics Bitmap Commands

When a bitmap is created, it is added to a list of bitmaps. Commands that create bitmaps return a pointer to the bitmap. The pointer is an index into the bitmap list. Your program works with the bitmap through the bitmap pointer.

If you want to draw the bitmap on the screen, you must add a graphical object to the Object List. The **Gr.bitmap.draw** command creates a graphical object that holds a pointer to the bitmap. Do not confuse the bitmap with the graphical object. You cannot use the Object Number to access the bitmap, and you cannot use the bitmap pointer to modify the graphical object.

Android devices limit the amount of memory available to your program. Bitmaps may use large blocks of memory, and so may exceed the application memory limit. If a command that creates a bitmap exceeds the limit, the bitmap is not created, and the command returns -1, an invalid bitmap pointer. Your program should test the bitmap pointer to find out if the bitmap was created. If the bitmap pointer is -1, you can call the **GetError$()** function to get information about the error.

If a command exceeds the memory limit, but BASIC! does not catch the out-of-memory condition, your program terminates with an error message displayed on the Console screen. If you return the Editor, a line will be highlighted near the one that exceeded the memory limit. It may not be exactly the right line.

Bitmaps use four bytes of memory for each pixel. The amount of memory used depends only on the width and height of the bitmap. The bitmap is not compressed. When you load a bitmap from a file, the file is usually in a compressed format, so the bitmap will usually be larger than the file.

The assigning of bitmap pointers has changed, because it is now possible to overwrite bitmaps. To have a unique bitmap pointer number, the value of the variable must be 0 before a command is executed. This is important when creating a bitmap for the first time. If the bitmap pointer variable has never been used before, its value is 0. If the value of the bitmap pointer variable is greater than 0 and the bitmap has been mapped to the internal bitmap list, that bitmap will be overwritten. You can also use members of an array as a bitmap pointer.

29.4.1 Gr.bitmap.clr

Syntax: Gr.bitmap.clr <bitmap_ptr_nexp> {, <paint_nexp>}

Fills a bitmap completely with transparency, but without destroying it. Anything already on the bitmap is cleared. There is no alpha blending.

It can obtain optional color from the paint specified by <paint_nexp>.

This command is approximately ten times faster than the combination of **Gr.bitmap.delete** and **Gr.bitmap.create**.

29.4.2 Gr.bitmap.create

Syntax: Gr.bitmap.create <bitmap_ptr_nvar>, width, height

Creates an empty bitmap of the specified width and height. The specified width and height may be greater than the size of the screen, if needed.

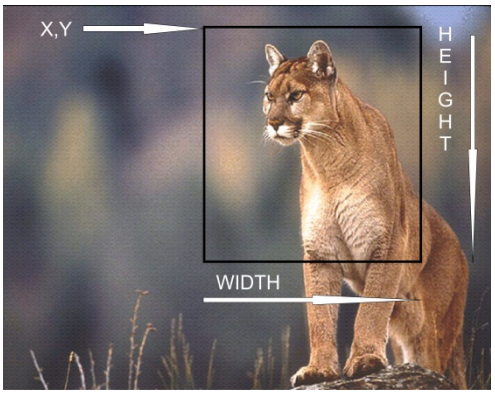
Returns a pointer to the created bitmap in the <bitmap_ptr_nvar> variable for use with the other **Gr.bitmap** commands. If there is not enough memory available to create the bitmap, the returned bitmap pointer is -1. Call **GetError$()** for information about the failure.

29.4.3 Gr.bitmap.crop

Syntax: Gr.bitmap.crop <new_bitmap_ptr_nvar>, <source_bitmap_ptr_nexp>, <x_nexp>, <y_nexp>, <width_nexp>, <height_nexp>

Creates a cropped copy of an existing source bitmap specified by <source_bitmap_ptr_nexp>. The source bitmap is unaffected; a rectangular section is copied into a new bitmap. A pointer to the new bitmap is returned in <new_bitmap_nvar>. If there is not enough memory available to create the new bitmap, the returned bitmap pointer is -1. Call **GetError$()** for information about the failure.

The <x_nexp>, <y_nexp> pair specifies the point within the source bitmap that the crop is to start at. The <width_nexp>, <height_nexp> pair defines the size of the rectangular region to crop.



This example shows how to check for out-of-bounds condition:

```
Gr.bitmap.size bp, wx, hy % 48, 48
x = 10
y = 20
width = 10
height = 15
If x <  0 | (x + width) > wx | width < 1 | y <  0 | (y + height) > hy | ~
 height < 1 Then
 Print "Error: The crop frame is out of bounds"
 End
EndIf
Gr.bitmap.crop newBp, bp, x, y
```

29.4.4 Gr.bitmap.delete

Syntax: Gr.bitmap.delete <bitmap_ptr_nexp>

Deletes an existing bitmap. The bitmap's memory is returned to the system.

This does not destroy any graphical object that points to the bitmap. If you do not **Gr.hide** such objects, or remove them from the Display List, you will get a run-time error from the next **Gr.render** command.

29.4.5 Gr.bitmap.draw

Syntax: Gr.bitmap.draw <object_ptr_nvar>, <bitmap_ptr_nexp>, x , y

Creates a graphical object that contains a bitmap and inserts the object into the Object List. The bitmap is specified by the bitmap pointer <bitmap_ptr_nexp>. The bitmap will be drawn with its upper left corner at the provided (x,y) coordinates. The command returns the Object List object number of the graphical object in the <object_ptr_nvar> variable. This object will not be visible until the **Gr.render** command is called.

The alpha value of the latest **Gr.color** will determine the transparency of the bitmap.

The **Gr.modify** parameters for **Gr.bitmap.draw** are "bitmap", "x" and "y".

29.4.6 Gr.bitmap.drawInto.end

Syntax: Gr.bitmap.drawInto.end {<legacy_mode_nexp>}

End the draw-into-bitmap mode. Subsequent draw commands will place the objects into the display list for rendering on the screen. If you wish to display the drawn-into bitmap on the screen, issue a **Gr.bitmap.draw** command for that bitmap.

For legacy reasons <legacy_mode_nexp> suppresses a runtime error, if **Gr.bitmap.drawInto.start** was not previously executed.

29.4.7 Gr.bitmap.drawInto.start

Syntax: Gr.bitmap.drawInto.start <bitmap_ptr_nexp>

Put BASIC! into the draw-into-bitmap mode.

All draw commands issued while in this mode draw directly into the bitmap. The objects drawn in this mode are not placed into the display list. The object number returned by the draw commands while in this mode is invalid and should not be used for any purpose including **Gr.modify**.

Note: Bitmaps loaded with the **Gr.bitmap.load** command cannot be changed with **Gr.bitmap.drawInto**. To draw into an image loaded from a file, first create an empty bitmap then draw the loaded bitmap into the empty bitmap.

29.4.8 Gr.bitmap.fill

Syntax: Gr.bitmap.fill <bitmap_ptr_nexp>, <x_nexp>, <y_nexp>

Change all of the points in an area of a bitmap to the current drawing color. The bitmap pointer parameter <bitmap_ptr_nexp> must specify an existing bitmap. The x and y parameters <x_nexp> and <y_nexp> must specify a point (x, y) in the bitmap. The area to color is a set of connected pixels all the same color. The area may be any shape, and the point (x, y) may be any point in the area.

This command reads actual bitmap pixel colors, so it is affected by the antialiasing setting. If antialiasing is on, the pixels at the edge of the colored area may not be re-colored correctly.

29.4.9 Gr.bitmap.filter

Syntax: Gr.bitmap.filter <new_bitmap_ptr_nvar>, <bitmap_ptr_nexp>, <bundl_ptr_nexp>

Processes a previously loaded bitmap (<bitmap_ptr_nexp>) by filters which are specified by the

<bundl_ptr_nexp> bundle, and creates a new bitmap <new_bitmap_ptr_nvar>. The old bitmap still exists; it is not deleted.

Table of Bundle Keys		
Key	**Value**	**Description**
_Filter	_Binary _BlackFilter _Blur _Brightness _ColorRotation _ColorScale _EdgeDetection _Engrave _Flip _GammaCorrection _Hue _Invert _MaskPoly _ModifyOrientation _PolyToPoly _Rotate _RoundCorners _Saturation _Skew _Smooth _SnowEffect _Sharpen _UseColorMatrix _UseConvolutionMatrix _Watermark (String)	Set the filter type. Always needed!
_Binary The result is a Bitmap with only white and black pixels.		
		Default values.
_BlackFilter Base on randomizing image pixels, enhance the noise of darkness. The algorithm is to generate a threshold number (0-255), if all R,G,B values of a pixel are less than the threshold, then set the pixel to black.		
		Default values. It is a pixel by pixel function, so it needs more time.
_Blur Also called Gaussian Blur Effect		
_Radius	0.1 to 25 (numeric)	Default is 15.
_Brightness see _ContrastBrightness		
_Value	From -255 to +255 (numeric)	Default is 0. The result is limited to the possible minimum or maximum.

Table of Bundle Keys		
Key	**Value**	**Description**
_ColorRotation		
Set the rotation on a color axis by the specified values.		
_Axis	_Red, _Green or _Blue (String)	Default is _Red.
_Degrees	0 to 360 (numeric)	Default is 0.
_ColorScale		
_AlphaScale	0 and positive numbers (numeric)	A value of 0 switches the color to off. 1 is identity and default. The result is limited to the possible maximum.
_RedScale	0 and positive numbers (numeric)	A value of 0 switches the color to off. 1 is identity and default. The result is limited to the possible maximum.
_GreenScale	0 and positive numbers (numeric)	A value of 0 switches the color to off. 1 is identity and default. The result is limited to the possible maximum.
_BlueScale	0 and positive numbers (numeric)	A value of 0 switches the color to off. 1 is identity and default. The result is limited to the possible maximum.
_Contrast		
_Value	0 and positive numbers (numeric)	A value of 0 sets the contrast to minimal. 1 is identity and default. The result is limited to the possible maximum.
_ContrastBrightness		
Contrast is the difference in luminance and/or color that makes an object (or its representation in an image or display) distinguishable. In visual perception of the real world, contrast is determined by the difference in the color and brightness of the object and other objects within the same field of view. The concept of brightness is rather simple, increasing/decreasing value of each R, G, B channel together. + By increasing: image results brighter. - By decreasing: image results darker.		
_BrightnessValue	From -255 to +255 (numeric)	Default is 0. The result is limited to the possible minimum or maximum.
_ContrastValue	0 and positive numbers (numeric)	A value of 0 sets the contrast to minimal. 1 is identity and default. The result is limited to the possible maximum.
_EdgeDetection		
_Level	_Low, _Medium or _High (String)	Default is _Medium. The speed could be faster, but the results of the faster Android Render Script are strange.
_Engrave		

Table of Bundle Keys		
Key	**Value**	**Description**
		Default values. It is a pixel by pixel function, so it needs more time.
_Hue		
Translates the colors within the color wheel by the specified angle. White, gray and black are not affected.		
_Degrees	0 to 360 (numeric)	Default is 0.
_Flip		
_Horizontal	0 or 1 (numeric)	Default is 0.
_Vertical	0 or 1 (numeric)	Default is 0.
_GammaCorrection		
_Gamma	0 to 8 (numeric)	1 is identity and default. Correction by a 1 / gamma value. Values below 1 provide darker results. Values greater than 1 provide brighter results. It is a pixel by pixel function, so it needs more time.
_Invert		
Inverts the bitmap from a positive into a negative or a negative into a positive one.		
		Default values.
_MaskPoly		
Masks the bitmap with a polygon.		
_SourcePoints	Array (numeric)	Default is {0,0,0,0,0,0,0,0} Array of polygon nodes positions as x,y pairs. You can use polygons with three or more nodes.
_Type	_In or _Out (String)	Default is _In. If the _Type is _In the area inside the polygon is returned as **transparent**. The argument _Out stands for the area outside.
_ModifyOrientation		
Modifies the orientation of the current bitmap specified by the EXIF of a bitmap given by an URL.		
_ImageUrl	(String)	
_PolyToPoly		
Crops from a source bitmap a part defined by a polygon with three or four nodes. The result is described by a destination polygon with the same number of nodes. The order of the nodes are beginning at the left top corner in a clockwise turned direction. Sometimes we get **in trouble,** if we use a polygon with **four** nodes and a small source bitmap. In this case try to scale the source bitmap to double size or more.		
_SourcePoints	Array (numeric)	Default is {0,0,0,0,0,0,0,0} Array of polygon nodes positions as x,y pairs. You can use polygons with three or four nodes.
_DestinationPoints	Array (numeric)	Default is {0,0,0,0,0,0,0,0} Array of polygon nodes positions as x,y

Table of Bundle Keys		
Key	**Value**	**Description**
		pairs. You can use polygons with three or four nodes.
_Mask	0 or 1 (numeric)	Default = 1 Masks the source bitmap by the _SourcePoints to get a transparent area outside the polygon.
_Debug	0 or 1 (numeric)	Debug results will be printed at the console.
_Rotate		
_Degrees	0 to 360 (numeric)	Default is 0. Keep in mind, that an angle <> 0, 180 perhaps also 90 and 270 (width equals high) will make the bitmap larger. Areas of the resulting four triangles are transparent.
_RoundCorners		
_Radius	0 and positive numbers (numeric)	Default is 0.
_Saturation		
_Value	0 and positive numbers (numeric)	A value of 0 maps the color to **gray-scale**. 1 is identity and default. Values > 1 **boost** the colors. The result is limited to the possible maximum.
_Sharpen		
_Level	_Low, _Medium or _High (String)	Default is _Medium.
_Skew		
_DeltaX	0 and positive numbers (numeric)	Default is 0. Sets the x size of the skew.
_DeltaY	0 and positive numbers (numeric)	Default is 0. Sets the y size of the skew.
_AtX	0 and positive numbers (numeric)	Default is 0. Sets the x position of the skew.
_AtY	0 and positive numbers (numeric)	Default is 0. Sets the y position of the skew.
_Smooth		
_Value		Default is 0.
_SnowEffect Opposite of Black Filter Sets all pixels having R,G,B values to the max of 255 when they are greater than threshold.		
		Default values. It is a pixel by pixel function, so it needs more time.
_UseColorMatrix		
_ColorMatrix	Example: Array.Load colorMatrix[],~ %[-1, 0, 0, 0, 255,~	

Table of Bundle Keys		
Key	**Value**	**Description**
	0, -1, 0, 0, 255,~ 0, 0, -1, 0, 255,~ 0, 0, 0, 1, 0 %] (numeric array)	
_UseConvolutionMatrix		
_ConvolutionMatrix	Example for _3x3 Matrix: Array.Load covolutionMatrix[],~ %[-0.15, -0.15, -0.15,~ -0.15, 2.2, -0.15,~ -0.15, -0.15, -0.15 %] (numeric array)	
_Type	_3x3 or _5x5 (String)	Default is _3x3.
_ColorSpace	_ARGB_8888 or _RGB_565 (String)	Default is _ARGB_8888. Use the _RGB_565 option with care, because there is no chance to handle an error!
_Watermark Gives your image a watermark.		
_Text	(String)	Default is "". Text of the watermark.
_Color	{Alpha,}Red,Green,Blue (comma delimited string) or _{Alpha,}ColorName ({comma delim.} string) or #{hn}hnhnhn (hex. string)	Default is "".
_AtX	0 and positive numbers (numeric)	Default is 0. Sets the x position of the watermark.
_AtY	0 and positive numbers (numeric)	Default is 0. Sets the y position of the watermark.
_Size	0 and positive numbers (numeric)	Default is 10. Font size of the watermark.
_Underline	0 or 1 (numeric)	Default is 0. If > 0 the watermark has an underline.

Example:

```
Gr.open "_White", 1, 1
Gr.bitmap.load bPtr1, "cartman.png"
Bundle.put bndPtr1, "_Filter", "_Hue"
counter = 0
Do
 Bundle.put bndPtr1, "_Degrees", counter
 Gr.bitmap.scale bPtr2, bPtr1, 600,600
 Gr.bitmap.filter bPtr3, bPtr2, bndPtr1
 Gr.bitmap.draw oPtr1, bPtr3, 250, 250
 Gr.render
 counter = counter + 5
```

```
Until counter = 365
```

29.4.10 Gr.bitmap.get.histogram

Syntax: Gr.bitmap.get.histogram <bitmap_ptr_nexp>, alpha[], red[], green[], blue[]

The bitmap pointer is specified by <bitmap_ptr_nexp>. This command returns the color channel specific histogram arrays alpha[], red[], green[] and blue[]. Each array contains the sum of pixels with the same intensity. If 47 pixel have the red intensity of 233, red[234] returns 47 (234 because BASIC! arrays are starting with an index of 1 instead of 0).

29.4.11 Gr.bitmap.get.pixarr

Syntax: Gr.bitmap.get.pixarr <bitmap_ptr_nexp>, alpha[], red[], green[], blue[]{, colorNumbers[]}

The bitmap pointer is specified by <bitmap_ptr_nexp>. This command returns the color channel specific pixel arrays alpha[], red[], green[] and blue[]. The optional colorNumbers[] returns an array of system color numbers. The order is column by column.

Keep in mind, that the first left top pixel point is 1, 1 instead of 0, 0, because the BASIC! Array index base is 1. The last right bottom pixel point is width, height.

Example of a bitmap with 5 pixels width and 2 pixels height:

y\x	1	2	3	4	5
1	0	200	40	60	80
2	100	30	50	70	90

[3,2] = 50

Example:

```
Gr.open
Gr.bitmap.load bPtr, "cartman.png"
Gr.bitmap.get.pixarr bPtr, alpha[], red[], green[], blue[]
Array.dims alpha[], dims[]
Debug.on
Debug.dump.array dims[]
```

29.4.12 Gr.bitmap.get.selected.pixarr

Syntax: Gr.bitmap.get.selected.pixarr <bitmap_ptr_nexp>, x[], y[], alpha[], red[], green[], blue[]{, colorNumbers[]}

The bitmap pointer is specified by <bitmap_ptr_nexp>. The command returns the color channel specific pixel arrays alpha[], red[], green[] and blue[] at the pixel points defined by the x[] and y[] arrays. The optional colorNumbers[] returns an array of system color numbers.

If a pixel point is outside of the bitmap, the color channels return -1. In case of colorNumbers[], Infinity will be returned. For example:

```
If colorNumbers[1] < VAL("Infinity") Then Print "Within"
```

Keep in mind that the top left pixel point is 0 (x[]), 0 (y[]). The last right bottom pixel point is width -1, height -1.

29.4.13 Gr.bitmap.load

Syntax: Gr.bitmap.load <bitmap_ptr_nvar>, <file_name_sexp>{{{{{{{, <wB_nexp>}, <hB_nexp>}, <cropX_nexp>}, <cropY_nexp>}, <cropW_nexp>}, <cropH_nexp>}, <bgColor_sexp>}

Creates a bitmap from the file specified in the <file_name_sexp> string expression. Returns a pointer to the created bitmap for use with other **Gr.bitmap** commands. If no bitmap is created, the returned bitmap pointer is -1. Call **GetError$()** for information about the failure. Some of the possible causes are:

- The file or resource does not exist.
- There is not enough memory available to create the bitmap.

Bitmap image files are assumed to be located in the "<pref base drive>/rfo-basic/data/" directory.

Note: You may include path fields in the file name. For example, "../../Cougar.jpg" would cause BASIC! to look for Cougar.jpg in the top level directory of the base drive, usually the SD card. "images/Kitty.png" would cause BASIC! to look in the images(d) sub-directory of the "/sdcard/rfo-basic/data/" ("/sdcard/rfo-basic/data/images/Kitty.png").

Note: Bitmaps loaded with this command cannot be changed with the **Gr.bitmap.drawInto** command. To draw into an image loaded from a file, first create an empty bitmap then draw the loaded bitmap into the empty bitmap.

Bitmap image files are assumed to be located as non source and non database files in the main data directory by default.

If a SVG (Scalable Vector Graphic) file is chosen the result based on the measurement in relation to 96 DPI. I.e. 5 cm /(2.54 cm/Inch) * 96 (Dots/Inch) = 189 Dots.

When rendering an SVG file, the Android "_Sans_Serif" font is default. Special fonts like Arial, Verdana etc. are not supported.

The following arguments are optional.

The border arguments <wB_nexp> and <hB_nexp> specify the borders of the result.

The bitmap is scaled inside these borders, so that a square is a square and not a rectangle.

<wB_nexp>	<hB_nexp>	Result
0 (default)	0 (default)	In the original resolution
> 0	0	Scaled to the given width
0	> 0	Scaled to the given height

Note that SVG files need to be converted large enough for the best quality.

The arguments <cropX_nexp>, <cropY_nexp>, <cropW_nexp>, <cropH_nexp> describe the position of the left (cropX) top (cropY) corner, and the size (cropW, cropH) of the part to cut out. The defaults are 0, 0, -1, -1. If cropW is -1, the right edge limits the cut out. If cropH is -1 the bottom edge limits the cut out. So the defaults return the full size.

To prevent memory faults, it is a good idea to crop at loading, because older Android versions limit the maximum images size for loading to ≤ 2048 x ≤ 2048 pixel ≤ 12 MB RAM.

See also the <file_name_sexp> extension of Gr.bitmap.size for more information.

Mainly for buttons in conjunction with icons, <bgColor_sexp> sets the background color by the Gr.Paint color notation.

Example:

```
Gr.bitmap.size "blocks.svg", width, height
! Crop the right bottom quarter of "blocks.svg" in its original resolution + a
new blue backgr.
Gr.bitmap.load bPtr, "blocks.svg", 0, 0, width/2, height/2, -1, -1, "_10,Blue"
! Try also "_Black" and  "_Red"
```

29.4.14 Gr.bitmap.save

Syntax: Gr.bitmap.save <bitmap_ptr_nvar>, <filename_sexp>{, <quality_nexp>}

Saves the specified bitmap to a file. The default path is "<pref base drive>/rfo-basic/data/".

The file will be saved as a JPEG file if the filename ends in ".jpg". The file will be saved as a PNG file if the filename ends in anything else (including ".png").

The range for <quality_nexp> is 0 to 100. The default is 50.

29.4.15 Gr.bitmap.scale

Syntax: Gr.bitmap.scale <new_bitmap_ptr_nvar>, <bitmap_ptr_nexp>, width, height {, <smoothing_lexp>}

Scales a previously loaded bitmap (<bitmap_ptr_nexp>) to the specified width and height and creates a new bitmap <new_bitmap_ptr_nvar>. The old bitmap still exists; it is not deleted. If there is not enough memory available to create the new bitmap, the returned bitmap pointer is -1. Call **GetError$()** for information about the failure.

Negative values for width and height will cause the image to be flipped left to right or upside down.

Neither the width value nor the height value may be zero.

Use the optional smoothing logical expression (<smoothing_lexp>) to request that the scaled image not be smoothed. If the expression is false (zero) then the image will not be smoothed. If the optional parameter is true (not zero) or not specified then the image will be smoothed.

29.4.16 Gr.bitmap.set.pixarr

Syntax: Gr.bitmap.set.pixarr <bitmap_ptr_nvar>, alpha[]{, red[], green[], blue[]}

Creates a bitmap by the bitmap pointer <bitmap_ptr_nvar> and sets the pixels by the color channel specific pixel arrays. The order is column by column specified by alpha[]. An alpha[] array dimensioned by [5,2] returns a bitmap with 5 pixels width and 2 pixels height. The defaults of the arrays red[], green[], blue[] are filled with 255. Values < 0 are changed to 0 and values > 255 to 255.

Example of a bitmap with 5 pixels width and 2 pixels height:

y\x	1	2	3	4	5
1	0	200	40	60	80
2	100	30	50	70	90

[3,2] = 50

Example:

```
! Bitmap 100*40 with random colored pixels
b = 100 : h = 40 : bS = b * h
Dim bSArray[bS]
```

```
Array.fill bSArray[], 255
Array.load d[], b, h % 100*40 = 4000
Array.to.dims bSArray[], d[], alpha[] % Only alpha[] dims need to be specified
Array.rnd red[], bS, 0, 255
Array.rnd green[], bS, 0, 255
Array.rnd blue[], bS, 0, 255
Gr.bitmap.set.pixarr nRndPtr, alpha[], red[], green[], blue[]
Gr.bitmap.draw rndPtr, nRndPtr, 200, 600
Gr.render
```

29.4.17 Gr.bitmap.size

Syntax: Gr.bitmap.size <bitmap_ptr_nexp>|<file_mame_sexp>, width, height

Return the pixel width and height of the bitmap pointed to by <bitmap_ptr_nexp> into the width and height variables.

Return the pixel width and height of the bitmap pointed to by <bitmap_ptr_nexp> or <file_name_sexp> into the width and height variables. SVG (Scalable Vector Graphic) file names ending with *.svg are also supported. In this case, the measurements are factored by 96 DPI; for example 5 cm / (2.54 cm/Inch) * 96 (Dots/Inch) = 189 Dots.

The advantage of <file_name_sexp> is the detection of the size before loading. This prevents memory faults, mostly on older devices.

The following should be noted about the memory requirements of bitmaps.

If you load bitmaps in their original size, the memory consumption can be very high, since even compressed Jpeg files are also decompressed when loading. The high resolutions of today's devices do the rest to make it a challenge.

Google therefore recommends only using or loading a reduced image or an image section.

That's why **Gr.bitmap.size** has an extension to get the size by filename directly, so that the bitmap size can be determined before loading. To do this, however, the bitmap must be loaded internally. To avoid this with large bitmaps, you can make a preselect by using **File.size**. For example, all compressed image files over 2 Mb can be reduced in size by using the **Gr.bitmap.load** command, **Gr.bitmap.load** has an extension that reduces the loaded image, or a section of the image.

29.4.18 Gr.get.bmpixel

Syntax: Gr.get.bmpixel <bitmap_ptr_nvar>, x, y, alpha, red, green, blue

Return the color data for the pixel of the specified bitmap at the specified x, y coordinate. The x and y values must not exceed the length or width of the bitmap.

29.5 Graphics Bundle Commands

29.5.1 Gr.bitmap.get

Syntax: Gr.bitmap.get <bundle_pointer_nexp>, <key_sexp>, <bitmap_ptr_nexp>

Gets a bitmap from a bundle.

Places the bitmap specified by the key string expression into the pointer specified bitmap. If the bundle does not exist or does not contain the requested key, the command generates a run-time error.

Example:

```
Gr.bitmap.get bptr,"Picture2", bitmapPtr
```

29.5.2 Gr.bitmap.put

Syntax: Gr.bitmap.put <bundle_pointer_nexp>, <key_sexp>, <bitmap_pointer_nexp>

Puts a bitmap into a bundle.

The bitmap will be placed into the specified bundle using the specified key. If the bundle does not exist, a new one may be created.

The type of the value is a bundle pointer.

Needs Graphic Mode.

Example:

```
Gr.bitmap.put bptr, "picture1", bitmapPtr
```

29.5.3 Gr.drawable.get

Syntax: Gr.drawable.get <bundle_pointer_nexp>, <key_sexp>, <drawable_ptr_nexp>

Gets a drawable **in the form of a bitmap** from a bundle.

Places the drawable specified by the key string expression into the pointer specified drawable. If the bundle does not exist or does not contain the requested key, the command generates a run-time error.

Example:

```
Gr.drawable.get bptr,"Picture2", drawablePtr
```

29.5.4 Gr.drawable.put

Syntax: Gr.drawable.put <bundle_pointer_nexp>, <key_sexp>, <drawable_pointer_nexp>

Puts a drawable **as a bitmap** into a bundle.

The drawable will be placed into the specified bundle using the specified key. If the bundle does not exist, a new one may be created.

The type of the value is a bundle pointer. The background references are cut off!

Needs Graphic Mode.

Example:

```
Gr.drawable.put bptr, "picture1", drawablePtr
```

29.6 Graphics Camera Commands

There are three ways to use the camera from BASIC!:

1) The device's built in Camera User Interface can be used to capture an image. This method provides access to all the image-capture features that you get when you execute the device's Camera application. The difference is the image bitmap is returned to BASIC! for manipulation by BASIC! The **Gr.camera.shoot** command implements this mode.
2) A picture can be taken automatically when the command is executed. This mode allows for

untended, time-sequenced image capture. The command provides for the setting the flash to on, off and auto. The **Gr.camera.autoShoot** command implements this mode.

3) The third mode is the **Gr.camera.manualShoot** command which is much like the autoshoot mode. The difference is that a live preview is provided and the image is not captured until the screen is touched.

All pictures are taken at full camera resolution and stored with 100% jpg quality as "<pref base drive>/rfo-basic/data/image.png".

All of these commands also return pointers to bitmaps. The bitmaps produced are scaled down by a factor of 4. You may end up generating several other bitmaps from these returned bitmaps. For example, you many need to scale the returned bitmap to get it to fit onto your screen. Any bitmaps that you are not going to draw and render should be deleted using **Gr.bitmap.delete** to avoid out-of-memory situations.

The Sample Program, f33_camera.bas, demonstrates all the modes of camera operations. It also provides examples of scaling the returned image to fit the screen, writing text on the image and deleting obsolete bitmaps.

The Sample Program, f34_remote_camera.bas, demonstrates remote image capture using two different Android devices.

29.6.1 Gr.camera.autoShoot

Syntax: Gr.camera.autoShoot <bm_ptr_nvar>{{{{{, <flash_ mode_nexp>}, <focus_mode_nexp>}, <orientation_nexp>} , <take_params_nexp>} , <file_name_sexp>}

An image is captured as soon as the command is executed. No user interaction is required. This command can be used for untended, time-sequence image captures.

The optional flash_mode numeric expression specifies the flash operation:

0	Auto Flash
1	Flash On
2	Flash Off
3	Torch
4	Red-eye

The default, if no parameter is given, is Auto Flash.

The optional focus_mode numeric expression specifies the camera focus:

0	Auto Focus
1	Fixed Focus
2	Focus at Infinity
3	Macro Focus (close-up)

The default, if no parameter is given, is Auto Focus.

If you want to specify a focus mode, you must also specify a flash mode.

If the <orientation_nexp> is not present or -1, the default orientation is Landscape.

If <take_params_nexp> is greater than 0 camera parameters are taken over:

0	No take over
1	parameters created with **Gr.camera.setparam** in the simple **GR.open** mode

2	parameters created with **Gr.camera.setparam** in the **GR.open** mode with hidden camera preview

<flash_ mode_nexp>, <focus_mode_nexp>, <orientation_nexp> and picture size are overwritten by the command settings and default settings.

The command also stores the captured image into the file, "<pref base drive>/rfo-basic/data/image.png" by default. If the optional <file_name_sexp> is specified with "", the default file path is used.
<bm_ptr_nvar> returns a bitmap, which only covers the screen in size, thus its width or its height will be a little larger.

Keep in mind, that the time gap between two Gr.camera.***Shoot commands should be round about 500 milliseconds. If a user input is necessary, a separate Pause command is not needed.

29.6.2 Gr.camera.directShoot

Syntax: Gr.camera.directShoot <bm_ptr_nvar>{{,<file_name_sexp>} <size_sexp>}

An image is captured as soon as the command is executed. <bm_ptr_nvar> returns a bitmap pointer.

The optional <file_name_sexp> can specify a file name to save the image.

By default, the bitmap is the same size as the screen. But The optional argument <size_sexp> can be used to scale it. It can be set to "_Max" or an actual size like "1920x1080".

If this command does not work well in your Android device, try using a **Gr.camera.***Shoot** command instead.

29.6.3 Gr.camera.flash

Syntax: Gr.camera.flash <Flash_mode_nval>

Sets the Flash mode of the current camera, but only if the camera was opened by **Gr.open**.

The <Flash_mode_nval> value specifies the flash operation:

Flash_mode_nval	Flash Operation
0	Auto Flash
1	Flash On
2	Flash Off
3	Torch
4	Red-eye

The default, if no parameter is given, is 2, Flash Off. If a failure occurs, <Flash_mode_nval> returns -1.

If no camera is running, you can use the flash like a torch. But this is only available in Graphics Mode.

Torch Example:

```
Gr.open 255,80,80,80, 1, -1, 1
on = 3
Gr.camera.flash on
Pause 5000
off = 2
Gr.camera.flash off
```

29.6.4 Gr.camera.focus

Syntax: Gr.camera.focus <Focus_mode_nval>

Sets the focus mode of the current camera, but only if the camera was opened by **Gr.open**.

The <Focus_mode_nval> numeric value specifies the camera focus:

<Focus_mode_nval>	Camera Focus
0	Auto Focus
1	Fixed Focus
2	Focus at Infinity
3	Macro Focus (close-up)
4	Continuous Picture
5	Continuous Video

The default, if no parameter is given, is Auto Focus and Continuous Picture. If a failure occurs, <Focus_mode_nval> returns -1.

29.6.5 Gr.camera.getParam

Syntax: Gr.camera.getParam <param_svar>

Returns the actual parameter of the currently selected camera. The supported camera parameters differ from one Android device to another.

Can only be used with Android 5+ and with Android 4 with limitations.

29.6.6 Gr.camera.manualShoot

Syntax: Gr.camera.manualShoot <bm_ptr_nvar>{{{{{, <flash_ mode_nexp> },
<focus_mode_nexp>} , <orientation_nexp>} , <take_params_nexp>} ,
<file_name_sexp>}

This command is much like **Gr.camera.autoshoot** except that a live preview is shown on the screen. The image will not be captured until the user taps the screen.

29.6.7 Gr.camera.select

Syntax: Gr.camera.select 1|2|...

Selects the Back (1) or Front(2) camera in devices with two cameras. The default camera is the back (opposite the screen) camera. Modern devices can have more than two cameras, but only the main rear camera and the front camera are supported.

If only one camera exists, then the default will be that camera. For example, if the device (such as the Nexus 7) only has a Front Camera then it will be the default camera. If the device does not have any installed camera apps, then there will be a run-time error message, "This device does not have a camera." In addition, a run-time error message will be shown if the device does not have the type of camera (front or back) selected.

29.6.8 Gr.camera.setParam

Syntax: Gr.camera.setParam <param_sexp>

Sets one ore more parameters of the currently selected camera. The supported camera parameters differ from one Android device to another.

Can only be used with Android 6+.

Example:

```
GR.CAMERA.GETPARAM p$
p$ = REPLACE$(p$, ";",";\n")
```

```
PRINT p$
s$ = "effect=mono;iso=ISO200" % The ; is the delimiter.
GR.CAMERA.SETPARAM s$, 1
```

29.6.9 Gr.camera.shoot

Syntax: Gr.camera.shoot <bm_ptr_nvar>{, 0, 0, 0, 0, <file_name_sexp>}

This command calls the device's built in camera user interface to take a picture. The image is returned to BASIC! as a bitmap pointed to by the <bm_ptr_nvar> numeric variable. If the camera interface does not, for some reason, take a picture, <bm_ptr_nvar> will be returned with a zero value.

The command also stores the captured image into the file, "<pref base drive>/rfo-basic/data/image.jpg(png)" as default if possible. If the optional <file_name_sexp> is specified with "", the default file path is used. <bm_ptr_nvar> returns a bitmap. It is the same size as the as the screen, it is not scaled. Otherwise, the bitmap is scaled in the manner that it only covers the screen in size, thus this is in the width or the height it is a little larger.

Many of the device camera interfaces will also store the captured images somewhere else in memory with a date coded filename. These images can be found with the gallery application. BASIC! is not able to prevent these extraneous files from being created.

Note: Some devices like the Nexus 7 do not come with a built in camera interface. If you have installed an aftermarket camera application then it will be called when executing this command. You can take pictures with the Nexus 7 (or similar devices) using the other commands even if you do not have camera application installed. If the device does not have any installed camera apps, then there will be a run-time error message, "This device does not have a camera."

29.6.10 Gr.camera.takeVideo

Syntax: Gr.camera.takeVideo <file_name_sexp> {{, <duration_limit_nexp> }, <size_limit_nexp>}

This command calls the device's built in camera user interface to take a video. The argument <file_name_sexp> sets the file name to store the video. The file name should end with ".mp4". If the file name is "", the file should be saved at the standard location, unfortunately, this is not always guaranteed.

The duration limit can be set in seconds with time <duration_limit_nexp>. This is not always guaranteed either.

The maximum file size can be set with <size_limit_nexp>, with the same restriction as before.

Example:

```
Gr.camera.takeVideo "video.mp4", 5, 12*1048*1048 %=12MB
File.root path$
!vv Shows the video based on an application to be selected
Browse "file://" + path$ + "/" + "video.mp4" % Absolute file path needed!
Pause 4000
!vv Stops playing after ~ 4 seconds in the default App for ".mp4" because "File
not found".
Browse "file://" + path$ + "/" + ".mp4"
```

29.6.11 Gr.camera.zoom

Syntax: Gr.camera.zoom <Zoom_factor_nvar>

Sets the zoom factor of the current camera, but only if the camera was opened by **Gr.open**.

If the zoom factor is larger as the camera's zoom maximum, <Zoom_factor_nval> returns the maximum

zoom factor. If a failure occurs, <Zoom_factor_nval> returns -1.

29.7 Graphics Drawable Commands

Beginning with Android 9.0 / Pie API 28, animated drawables are supported.

Unfortunately, file access to all types of animated drawables was not back-ported for earlier versions, until now.

Drawable is a parent type for drawing and animating graphics.

Child types are bitmap or vector graphics, animated or fixed.

Supported image file types are BMP, PNG, JPEG, WEBP, GIF and HEIF.

If the encoded image is an animated GIF or WEBP, the animation has to be started by the **GR.drawable.start** command.

Up to and including Android 8.1 / Oreo API 28, the image file types BMP, PNG, JPEG and GIF are supported.

But it is not possible to convert a simple bitmap to an animated GIF.

When a drawable is created, it is added to a list of drawables. Commands that create drawables return a pointer to the drawable. The pointer is an index into the drawable list. Your program works with the drawable through the drawable pointer.

If you want to draw a drawable on the screen, you must add a graphical object to the Object List. The **Gr.drawable.draw** command creates a graphical object that holds a pointer to the drawable. Do not confuse the drawable with the graphical object. You cannot use the Object Number to access the drawable, and you cannot use the drawable pointer to modify the graphical object.

Android devices limit the amount of memory available to your program. Drawables may use large blocks of memory, and may exceed the application memory limit. If a command that creates a drawable exceeds the limit, the drawable is not created, and the command returns -1, an invalid drawable pointer. Your program should test the drawable pointer to find out if the drawable was created. If the drawable pointer is -1, you can call the **GetError$()** function to get information about the error.

If a command exceeds the memory limit, but BASIC! does not catch the out-of-memory condition, your program terminates with an error message displayed on the Console screen. If you return to the Editor, a line will be highlighted near the one that exceeded the memory limit. It may not be exactly the right line.

Drawables can use up to four bytes of memory for each pixel. The amount of memory used depends mainly on the width and height of the drawable. The drawable is not compressed. When you load a drawable from a file, the file is usually in a compressed format, so the drawable will usually be larger than the file.

The counting of drawable pointers was changed because, if possible, drawables will be overwritten. To automatically obtain a new unique drawable pointer number, the value of the variable should be set to 0 before executing the command. This is important when creating a drawable for the first time. You can also use array members for a drawable pointers. If a numeric variable, or an array element has never been used before, its value is 0.

If the value of the drawable pointer variable is greater than 0 and has been mapped to the internal drawable list, that drawable will be overwritten.

29.7.1 Gr.drawable.delete

Syntax: Gr.drawable.delete <drawable_ptr_nexp>

Deletes an existing drawable. The drawable's memory is returned to the system.

This does not destroy any graphical object that points to the drawable. If you do not **Gr.hide** such objects, or remove them from the Display List, you will get a run-time error from the next **Gr.render** command.

29.7.2 Gr.drawable.draw

Syntax: Gr.drawable.draw <object_ptr_nvar>, <drawable_ptr_nexp>, left, top, right, bottom

Creates a graphical object that contains a drawable and inserts the object into the Object List. The drawable is specified by the drawable pointer <drawable_ptr_nexp>. The drawable will be drawn within the bounds of the parameters. The command returns the Object List object number of the graphical object in the <object_ptr_nvar> variable. This object will not be visible until the **Gr.render** command is called.

The alpha value of the latest **Gr.color** will determine the transparency of the drawable.

The **Gr.modify** parameters for **Gr.drawable.draw** are: "drawable", "left", "top", "right" and "bottom".

The use of borders instead of coordinates is due to the ability to resize the images more easily.

29.7.3 Gr.drawable.fromBitmap

Syntax: Gr.drawable.fromBitmap <drawable_ptr_nvar>, <bitmap_ptr_nexp>

Creates a drawable from a given bitmap specified by the bitmap pointer <bitmap_ptr_nexp>. Returns a pointer to the created drawable for use with other **Gr.drawable** commands.

Note: The bitmap referenced by <bitmap_ptr_nexp> must not be deleted, or subsequent drawable commands might not work.

29.7.4 Gr.drawable.load

Syntax: Gr.drawable.load <drawable_ptr_nvar>, <file_name_sexp>

Creates a drawable from the file specified in the file_name string expression. Returns a pointer to the created drawable for use with other **Gr.drawable** commands. If no drawable is created, the returned drawable pointer is -1. Call **GetError$()** for information about the failure. Some of the possible causes are:

- The file or resource does not exist.
- There is not enough memory available to create the drawable.

Drawable image files are assumed to be located in the "<pref base drive>/rfo-basic/data/" directory.

Note: Drawables loaded with this command cannot be changed directly. To draw into an image loaded from a file, first create an empty bitmap, then draw the loaded drawable into the empty bitmap with the **Gr.bitmap.drawinto.start** command.

29.7.5 Gr.drawable.start

Syntax: Gr.drawable.start <drawable_ptr_nvar>

Starts the animation of the drawable <drawable_ptr_nvar> if possible. A subsequent **Gr.render** is needed to start.

29.7.6 Gr.drawable.stop

Syntax: Gr.drawable.stop <drawable_ptr_nvar>

Stops the animation of the drawable <drawable_ptr_nvar> if possible. A subsequent Gr.render is needed to stop.

29.8 Graphics Groups

You can put graphical objects into groups. A group is a list of graphical objects. When you perform certain operation on a group, the operation is performed on each object in the group.

You group graphical objects by creating a Group Object on the Display List. You use the group by putting its object number in a graphics command where you would use any other graphical object number.

In this version of BASIC!, you can use a Group Object in these commands:

- **Gr.move**: moves all of the objects by the same x and y amounts
- **Gr.hide**: hides all of the objects
- **Gr.show**: shows (unhides) all of the objects
- **Gr.show.toggle**: any objects that are showing will be hidden, and any objects that are hidden will be shown.

You use graphics commands to act on the objects in the group's list. You use the **List** commands to act on the list: add objects, count the objects, clear the list, and so on.

Try running this example. Watch as the top circle moves to the right, then the top two, and finally the top three, as circles are added one-by-one to the list attached to the group.

```
Gr.open ,,,,,1
Gr.color ,255,0,0,2
Gr.circle c1,100,100,40
Gr.circle c2,100,200,40
Gr.circle c3,100,300,40
Gr.circle c4,100,400,40
Gr.render
Pause 1000                      % draw four red circles

Gr.group g, c1                  % create a group with one circle
Gr.get.value g, "list", gList   % get the group's list of objects
Gr.move g, 0, 50                % move whole group 0 up/down, 50 right
Gr.render                       % only one circle moves
Pause 1000

List.add gList, c2              % add another circle to the group's list
Gr.move g, 0, 50
Gr.render                       % two circles move
Pause 1000

List.add gList, c2              % add another circle to the group's list
Gr.move g, 0, 50                % three circles move
Gr.render
Pause 1000

Gr.close
```

29.8.1 Gr.group

Syntax: Gr.group <object_number_nvar>{, <obj_nexp>}...

Creates a group of graphical objects. All of the numeric expressions <obj_nexp> must evaluate to valid graphical object numbers. The object numbers are put in a list and attached to the group.

The <object_number_nvar> returns the Object List object number for the group.

The **Gr.modify** parameter is "list".

29.8.2 Gr.group.getDL

Syntax: Gr.group.getDL <object_number_nvar>

Creates a group from the current Display List. The Display List is copied to a new list that is attached to the group.

The <object_number_nvar> returns the Object List object number for the group.

The **Gr.modify** parameter is "list".

29.8.3 Gr.group.list

Syntax: Gr.group.list <object_number_nvar>, <list_ptr_nexp>

Creates a group from a list of graphical objects. The List is assumed to contain valid graphical object numbers, but it is not checked. The list is simply attached to the group.

The list pointer parameter <list_ptr_nexp> is optional. If you provide an expression that evaluates to a valid List pointer, the List that the pointer addresses supplies the graphical objects that are put in the group. Otherwise, the group is empty. If you provide a numeric variable that does not already point to a list, the variable is set to point to the group's empty list.

The <object_number_nvar> returns the Object List object number for the group.

The **Gr.modify** parameter is "list".

29.8.4 Gr.group.newDL

Syntax: Gr.group.newDL <object_number_nvar>

Replaces the existing Display List with a new list read from the specified group.

The <object_number_nvar> returns the Object List object number for the group.

The **Gr.modify** parameter is "list".

29.9 Graphics Paint Commands

29.9.1 Gr.paint.copy

Syntax: Gr.paint.copy {{<src_nexp>}{, <dst_nexp>}}

Copy the Paint object at the source pointer <src_nexp> to the destination pointer <dst_next>.

Both parameters are optional. If you wish to specify a destination, you must include a comma, whether or not you specify a source. If either parameter is omitted, or if its value is -1, the current Paint is used.

The Paint already at the destination pointer is replaced. If the destination is the current Paint, a newly-created paint becomes the current Paint.

This command has four forms, depending on which parameters are present:

```
Gr.paint.copy        % Duplicate the current Paint
Gr.paint.copy m      % Copy Paint m to the current Paint
Gr.paint.copy , n    % Overwrite Paint n so it is the same as the current Paint
Gr.paint.copy m, n   % Overwrite Paint n so it is the same as Paint m
```

29.9.2 Gr.paint.get

Syntax: Gr.paint.get <object_ptr_nvar>

Gets a pointer (<object_ptr_nvar>) to the last created Paint object. For information about Paint objects, see the section Graphics → Introduction → Paints.

This pointer can be used to change the Paint object associated with a draw object by means of the **Gr.modify** command. The **Gr.modify** parameter is "paint".

If you want to modify any of the paint characteristics of an object then you will need to create a current Paint object with those parameters changed. For example:

```
Gr.color 255,0,255,0,0
Gr.text.size 20
Gr.text.align 2
Gr.paint.get the_paint
Gr.modify shot, "paint", the_paint
```

changes the current text size and alignment as well as the color.

29.9.3 Gr.paint.list

Syntax: Gr.paint.list <paintPointerList_nexp>, <colorStringList_nexp>

Returns a Paint pointer list given by colors from a list of color strings. Note that all Paints will be newly created, so use this command as less as possible. Other current Paint options without the color are overwritten.

See **Gr.color** for color text definitions.

Example:

```
List.create n, paintPtrs
List.create s, colorList
For i = 240 To 0 STEP -1
  List.add colorList, "_HSV" + INT$(i) % Uses the Hue color wheel
Next
! Return a Paint color list with the FEM (Finite element method) tension colors
! from index = 1 (blue) lowest tension, perhaps -120 N/mm²
! index = 121 (green) middle tension, perhaps 0 N/mm²
! to index = 241 (red) highest tension, perhaps 120 n/mm²
Gr.paint.get memPaint % Saves the current Paint
Gr.paint.list paintPtrs, colorList % Returns list of Paint pointers
Gr.paint.set memPaint % Load the last Paint created before
```

29.9.4 Gr.paint.reset

Syntax: Gr.paint.reset {<nexp>}

Force the specified Paint to default settings:

```
Color opaque black (255, 0, 0, 0)
Antialias ON
```

```
Style FILL (0)
Minimum stroke width (0.0)
Stroke Cap (0)  → "_Butt"
Xfermode (-1)
```

The parameter is optional. If the parameter is omitted or set to -1, a new current Paint is created with default settings.

Note that all **Gr.text** settings will also be reset because they are Paint settings too.

29.9.5 Gr.paint.set

Syntax: Gr.paint.set <bundle_nexp>{, <paint_nexp>}

Creates a new, or overwrites an existing, Paint with arguments provided by the given bundle <bundle_nexp>.

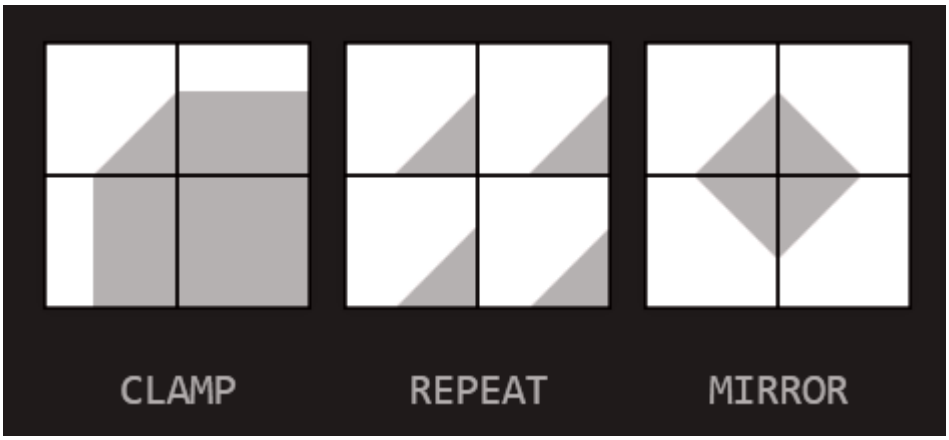
The optional <paint_nexp> selects the Paint definition to overwrite. If <paint_nexp> is -1, a new Paint definition is created. The default is 0 which uses the current Paint.

Table of Bundle Keys			
	Key	**Value**	**Description**
	_TextAlign	_Left, _Center or _Right	Align the text relative to the (x,y) coordinates given in the Gr.text.draw command.
	_TextBold	_On or _Off	
	_TextSkew	(numeric)	Set the paint's horizontal skew factor for text. The default value is 0. For approximating oblique text, use values around -0.25.
	_TextSize	(numeric)	Sets the text size.
	_TextScaleX	(numeric)	Set the paint's horizontal scale factor for text. The default value is 1.0. Values > 1.0 will stretch the text wider. Values < 1.0 will stretch the text narrower.
	_TextLetterSpace	(numeric)	Set the paint's letter-spacing for text. The default value is 0. The value is in 'EM' units. Typical values for slight expansion will be around 0.05. Negative values tighten text. Only Android 5+
	_TextWordSpace	(numeric)	Set the paint's extra word-spacing for text. Increases the white space width between words with the given amount of pixels. The default value is 0. Only Android 5+
	Font, style and typeface are also handled by Paint pointers. But please use Gr.text.setfont and Gr.text.typeface **behind Gr.paint.set.**		

	_Color	{Alpha,}Red,Green,Blue (comma delimited string) or _{Alpha,}ColorName ({comma delim.} string) or #{hn}hnhnhn (hex. string)	Sets the color. Default is "255,0,0,0" or "_Black", "_HSV240" or "#ff000000" See also Gr.color
	_Antialias	_On or _Off	Default is "_On". See also Gr.set.antialias
	_Style	_Fill, _Stroke or _Fill&Stroke	Default is "_Fill". See also Gr.color
	_StrokeWidth	(numeric)	Sets the stroke width. Default is 0.
	_StrokeCap	_CapButt, _CapRound or _CapSquare	Sets the line caps. Default is "_CapButt". See also Gr.set.cap
	_DashPathEffect	_On or _Off	See also Gr.set.dashpatheffect
	_PathPattern	list pointer (numeric)	The intervals list must contain an even number of entries (≥ 2), with the even indices specifying the "on" intervals, and the odd indices specifying the "off" intervals.
	_Phase	(numeric)	Phase is an offset into the intervals list (modifies the sum of all of the intervals).
	_Xfermode	_Normal, _Clear, _Src, _Dst, _Src_Over, _Dst_Over, _Src_In, _Dst_In, _Src_Out, _Dst_Out, _Src_Atop, _Dst_Atop, _Xor, _Darken, _Lighten, _Multiply, _Screen, _Add or _Overlay	Sets the Porter-Duff Compositing and Blend Modes. Default is "_Normal". See also Gr.color
	_Shader	_LinearGradient, _Pattern, _RadialGradient or _SweepGradient (String)	Creates a Shader
	_LinearGradient		
	_X0	(numeric)	Default is 0. Upper left corner in X direction
	_Y0	(numeric)	Default is 0. Upper left corner in Y direction
	_X1	(numeric)	Default is 0. Lower right corner in X direction
	_Y1	(numeric)	Default is 0. Lower right corner in Y direction

	_ColorArray	Array of {Alpha,}Red,Green,Blue (comma delimited string) or _{Alpha,}ColorName ({comma delim.} string) or #{hn}hnhnhn (hex. string)	Default is {"_Black", "_White"} An array of colors that creates the gradient from the top to the bottom.
	_PositionArray	Array (numeric)	Default is none. Array of positions
	_TileMode	_Clamp, _Mirror or _Repeat (String)	Default is "_Mirror". Sets the tile mode
	_Degree	(numeric)	Default is 0. Turns the Shader clockwise.
	_Pattern (Bitmap Shader)		
	_Bitmap	Bitmap object number (numeric)	Source bitmap
	_Degree	(numeric)	Default is 0. Turns the Shader clockwise.
	_TileModeX	_Clamp, _Mirror or _Repeat (String)	Default is "_Repeat". Sets the tile mode in X direction
	_TileModeY	_Clamp, _Mirror or _Repeat (String)	Default is "_Repeat". Sets the tile mode in Y direction
	_RadialGradient		
	_CenterX	(numeric)	Default is 0. Center point in X direction
	_CenterY	(numeric)	Default is 0. Center point in Y direction
	_Radius	> 0 (numeric)	
	_ColorArray	Array of {Alpha,}Red,Green,Blue (comma delimited string) or _{Alpha,}ColorName ({comma delim.} string) or #{hn}hnhnhn (hex. string)	Default is {"_Black", "_Withe"}
	_StopArray	Array (numeric)	Default is none. Array of stop points
	_TileMode	_Clamp, _Mirror or _Repeat (String)	Default is "_Mirror". Sets the tile mode
	_Degree	(numeric)	Default is 0. Turns the Shader clockwise.
	_SweepGradient		
	_CenterX	(numeric)	Default is 0. Center point in X direction
	_CenterY	(numeric)	Default is 0. Center point in Y direction

	_ColorArray	Array of {Alpha,}Red,Green,Blue (comma delimited string) or _{Alpha,}ColorName ({comma delim.} string) or #{hn}hnhnhn (hex. string)	Default is {"_Black", "_Withe"}
	_StopArray	Array (numeric)	Default is none. Array of stop points
	_Degree	(numeric)	Default is 0. Turns the Shader clockwise.



Tile Modes (Source: http://chiuki.github.io/android-shaders-filters/#/8)

Example:

```
Bundle.put bundlePtr2, "_Shader", "_Pattern"
Gr.bitmap.load sp1, "cartman.png"
Gr.bitmap.put bundlePtr2, "_Bitmap", sp1
Bundle.put bundlePtr2, "_Degree", 30
Gr.paint.set bundlePtr2
```

29.10 Graphics Path Commands

29.10.1 Gr.path

Syntax: Gr.path <obj_nvar>, <list_pointer_nvar> {, <x_nexp>, <y_nexp>}

Creates a path object defined by a list of strings with the system values "_MoveTo", "_LineTo", "_QuadTo" (Quadratic Bezier Curve), "_CubicTo" (Cubic Bezier Curve), "_ArcSegTo", "_ArcTo", "_Close" and "_End". The path will be located within the bounds of the parameters. The x and y parameters can be set to move the path. The path will or will not be filled depending upon the **Gr.color** style parameter. The <obj_nvar> returns the Object List object number for this rectangle. This object will not be visible until the **Gr.render** command is called.

The **Gr.modify** parameters for **Gr.path** are: "x", "y", "list", "paint" and "alpha".

Example:

```
Gr.open "_White", 0, 1
Gr.color "_Blue", 0
List.create S, pathDraft1
cPx = 250 : cPy = 250 % Center Points
R4 = 230
Angle1 = 30
Angle2 = 45
Angle3 = 60
! The following coordinates are relative values according
! to the optional placing values.
List.add pathDraft1, ~
 !"_MoveTo"~  %The first _MoveTo call is like GR.poly included.
 Str$(cPx), Str$(cPy+R4) ~ %Point 1
 !!b1
  Set the beginning of the next contour to the point (x1,y1).
  Moving like a pen in a pen plotter.
  Parameters:
  x1      The x-coordinate of the start of a new contour
  y1      The y-coordinate of the start of a new contour
 !!e1
 "_LineTo" ~
 Str$(cPx-R4*Cos(Angle2*Pi()/180)),Str$(cPy+R4*Sin(Angle2*Pi()/180)) ~ %Point 2
 !!b2
  Add a line from the last point {(x1,y1)} to the specified point (x2,y2).
  If no _MoveTo call has been made for this contour, the first point is
  automatically set to (0,0).
  Parameters:
  x2       The x-coordinate of the end of a line
  y2       The y-coordinate of the end of a line
 !!e2
 "_LineTo" ~
 Str$(cPx-R4), Str$(cPy) ~ %Point 2
 !!b3
  Add a quadratic bezier curve from the last point {(x1,y1)}, approaching
  control point (x2,y2), and ending at (x3,y3). If no _MoveTo or _LineTo call
  has been made for this contour, the first point is automatically set to (0,0)
  relative to the placing coordinates.
  Parameters:
  x2       The x-coordinate of the control point on a quadratic curve
  y2       The y-coordinate of the control point on a quadratic curve
  x3       The x-coordinate of the end point on a quadratic curve
  y3       The y-coordinate of the end point on a quadratic curve
 !!e3
 "_QuadTo" ~ %Quadratic Bezier Curve
 Str$(cPx-R4*Cos(Angle2*Pi()/180)), Str$(cPy-R4*Sin(Angle2*Pi()/180)) ~ %Point 2
 Str$(cPx), Str$(cPy-R4) ~ %Point 3
 !!b4
  Add a cubic bezier curve from the last point {(x1,y1)}, approaching control
  points (x2,y2) and (x3,y3), and ending at (x4,y4). If no _MoveTo or _LineTo
  call has been made for this contour, the first point is automatically set to
  (0,0) relative to the placing coordinates.
  Parameters:
  x2       The x-coordinate of the 1st control point on a cubic curve
  y2       The y-coordinate of the 1st control point on a cubic curve
  x3       The x-coordinate of the 2nd control point on a cubic curve
  y3       The y-coordinate of the 2nd control point on a cubic curve
  x4       The x-coordinate of the end point on a cubic curve
  y4       The y-coordinate of the end point on a cubic curve
 !!e4
 "_CubicTo" ~ %Cubic Bezier Curve
 Str$(cPx+R4*Cos(Angle3*Pi()/180)), Str$(cPy-R4*Sin(Angle3*Pi()/180)) ~ %Point 2
 Str$(cPx+R4*Cos(Angle1*Pi()/180)), Str$(cPy-R4*Sin(Angle1*Pi()/180)) ~ %Point 3
 Str$(cPx+R4), Str$(cPy) ~ %Point 4
 !!b5
  Append the specified arc to the path as a new contour. If the start of the
  path is different from the path's current last point, then an automatic
  _LineTo is added to connect the current contour to the start of the arc.
  However, if the path is empty, then we call _MoveTo with the first point of
  the arc.
```

```
 Parameters:
 Left, Top, Right, Bottom  The bounds of oval defining shape and size of arc.
 StartAngle                Starting angle (in degrees) where the arc begins
 SweepAngle                Sweep angle (in degrees) measured clockwise(!).
!!e5
"_ArcTo" ~
Str$(cPx), Str$(cPy-R4/4) ~ %Left, Top
Str$(cPx+R4), Str$(cPy+R4/4) ~ %Right, Bottom
Str$(0), Str$(180) ~ %StartAngle, SweepAngle
!!b6
 Append an arc to the path as a new contour. The arc is the specified by three
 coordinates. The start point is the last one of the path. The second and the
 third are the next coordinates.
 A line from coordinates (1) to coordinates (3) is drawn if coordinates (1),
 coordinates (2) and coordinates (3) are collinear, in this case also via (2);
 coordinates (1) and coordinates (2) are equal;
 coordinates (2) and coordinates (3) are equal or
 coordinates (3) are missed, in this case to (2).
 However, if the path is empty, then we call _MoveTo with the first point of
 the arc.
 Parameters:
 x2       The x-coordinate of the 1st control point on a cubic curve
 y2       The y-coordinate of the 1st control point on a cubic curve
 x3       The x-coordinate of the 2nd control point on a cubic curve
 y3       The y-coordinate of the 2nd control point on a cubic curve
!!e6
"_ArcSegTo" ~
Str$(cPx-R4/4), Str$(cPy-R4/4) ~ % Coordinates of the control point on an arc
Str$(cPx-R4/2), Str$(cPy) ~ %Coordinates of the end point on an arc
!!b7
Close the current contour. If the current point is not equal to the first
 point of the contour, a line segment is automatically added.
!!e7
!"_Close" ~
!!b8
 After _End the following items are ignored.
!!e8
"_End"

!!b9
 Gr.path <obj_nvar>, list_pointer {x, y}
 Parameters:
 obj_nvar      Object number
 list_pointer  List pointer of the list with curve type calls and coordinates
 {x, y}        Placing coordinates
!!e9
Gr.path  gfirst, pathDraft1,  20, 0

Gr.color "_Red"
Gr.circle cir, cPx + 20, cPy, R4

Gr.render
```

29.10.2 Gr.set.dashPathEffect

Syntax: Gr.set.dashPathEffect {<intervals_list_ptr_nvar>{, <phase_nexp>}}

Path Pattern:

The intervals list must contain an even number of entries (≥ 2), with the even indices specifying the "on" intervals, and the odd indices specifying the "off" intervals.

Phase:

Phase is an offset into the intervals list (modifies the sum of all of the intervals).

The intervals list controls the length of the dashes.

The **Gr.set.stroke** command controls the thickness of the dashes.

Note: This path effect only affects drawing when the **Gr.color**'s style parameter is set to 0 (STROKE) or 2 (STROKE_AND_FILL). It is ignored if the drawing is done with style = 1 (FILL).

A simple **Gr.set.dashPathEffect** without any arguments clears the effect to a full line.

Example:

```
scale = 2
List.create N, pathPatternType1
List.add pathPatternType1, 10 * scale, 7 * scale, 3 * scale, 7 * scale
Gr.set.dashPathEffect pathPatternType1, 5 * scale
```

29.11 Graphics Rotate Commands

These commands put graphics objects on the Display List like the GR drawing commands, but they don't draw anything. Instead they work as markers in the list. When the renderer sees the start marker, it temporarily rotates its coordinate system. The end marker tells the renderer to restore the coordinate system to normal.

The effect is to rotate or move any objects drawn after **Gr.rotate.start** and before **Gr.rotate.end**.

As with any graphics object, the rotate parameters may be changed with **Gr.modify**. At the next **Gr.render**, the rotated objects will be redrawn in their new positions.

29.11.1 Gr.rotate.end

Syntax: Gr.rotate.end {<obj_nvar>}

Ends the rotated drawing of objects. Objects created after this command will not be rotated.

The optional <obj_nvar> will contain the Object List object number of the **Gr.rotate.end** object. If you are going to use rotated objects in the array for **Gr.NewDL** then you will need to include the **Gr.rotate.start** and **Gr.rotate.end** objects.

29.11.2 Gr.rotate.start

Syntax: Gr.rotate.start angle, x, y{,<obj_nvar>}

Any objects drawn between the **Gr.rotate.start** and **Gr.rotate.end** will be rotated at the specified angle, in degrees, around the specified (x, y) point. If the angle is positive, objects are rotated clockwise.

The optional <obj_nvar> will contain the Object List object number of the **Gr.rotate.start** object. If you are going to use rotated objects in the array for **Gr.NewDL** then you will need to include the **Gr.rotate.start** and **Gr.rotate.end** objects.

The **Gr.modify** parameters for **Gr.rotate.start** are "angle", "x" and "y".

Gr.rotate.start must be eventually followed by **Gr.rotate.end** or you will not get the expected results.

29.12 Graphics Text Commands

Gr.text.draw is the only text command that creates a graphical object. The other text commands set

attributes of text yet to be drawn or report measurements of text.

Each command that sets a text attribute (**Gr.text.align**, **Gr.text.bold**, **Gr.text.size**, **Gr.text.skew**, **Gr.text.strike**, **Gr.text.underline**, **Gr.text.setFont** and **Gr.text.typeFace**) has an optional "Paint pointer" parameter that may be used to specify a Paint object to modify. Normally this parameter is omitted, and the command sets a text attribute for all text objects drawn after the command is executed. For using the Paint pointer, see "Paints *Advanced Usage*".

Gr.text.height returns information about how text would be drawn. It does not measure a drawn text object, but uses information from the current Paint.

Gr.text.width and **Gr.get.textBounds** commands can return information about how text would be drawn or how a text object was actually drawn. If used to measure the text of a string expression, they use information from the current Paint. If used to measure the text of a text object that was already drawn, they use information from the text object.

29.12.1 Gr.get.textBounds

Syntax: Gr.get.textBounds <exp>, left, top, right, bottom

Gets the boundary rectangle of a string as it would be drawn on the screen. The returned coordinate values give you the dimensions of the bounding rectangle but not its location.

The parameter <exp> follows the same rules as **Gr.text.width** to get a text string and the text attributes (typeface, size, and style) used to measure the string.

The coordinates of the rectangle are reported as if **Gr.text.draw** positioned your text at **0,0**. You get the actual boundaries of a text object by adding the textbounds offsets to the actual **x,y** coordinates of the **Gr.text.draw** command.

In typeface terminology, the coordinate values are offsets from the beginning of the baseline of the text (see **Gr.text.draw** and **Gr.text.height** for more explanation of the lines that define where text is drawn). This is why the value returned for "top" is always a negative number.

If this is confusing, try running this example:

```
Gr.open ,,,,,,-1
Gr.color 255, 255, 0, 0, 0
Gr.text.size 40
Gr.text.align 1
s$ = "This is only a test"
Gr.get.textBounds s$, l, t, r, b
Print l, t, r, b
x = 10 : y = 50
Gr.rect rct, l + x, t + y, r + x, b + y
Gr.text.draw txt, x, y, s$
Gr.render
Pause 5000
```

29.12.2 Gr.text.align

Syntax: Gr.text.align {{<type_nexp>}{,<paint_nexp>}}

Align the text relative to the (x,y) coordinates given in the **Gr.text.draw** command.

type values: 1 = Left, 2 = Center, 3 = Right.

You may use the optional Paint pointer parameter <paint_nexp> to specify a Paint object to modify. Normally this parameter is omitted. See **Gr.color**, *Advanced usage* for more information.

29.12.3 Gr.text.bold

Syntax: Gr.text.bold {{<lexp>}{,<paint_nexp>}}

Turns bold on or off on text objects drawn after this command is issued:

- If the value of the bold parameter <lexp> is false (0), text bold is turned off.
- If the parameter value is true (not zero), makes text appear bold.
- If the parameter is omitted, the bold setting is toggled.

If the color fill parameter is 0, only the outline of the bold text will be shown. If fill <>0, the text outline will be filled.

This is a "fake bold", simulated by graphical techniques. It may not look the same as text drawn after setting "Bold" style with **Gr.text.setFont** or **Gr.text.typeFace**.

You may use the optional Paint pointer parameter <paint_nexp> to specify a Paint object to modify. Normally this parameter is omitted. See **Gr.color**, *Advanced usage* for more information.

29.12.4 Gr.text.draw

Syntax: Gr.text.draw <object_number_nvar>, <x_nexp>, <y_nexp>, <text_object_sexp>

Creates and inserts a text object (<text_object_sexp>) into the Display List. The text object will use the latest color and text preparatory commands. The <object_number_nvar> returns the Display List object number for this text. This object will not be visible until the **Gr.render** command is called.

Gr.text.draw positions the text so the bodies of the characters sit on a line called the baseline. The tails of letters like "y" hang below the baseline.The **y** value <y_nexp> sets the location of this baseline.

The **Gr.text.height** command tells you the locations of the various lines used to draw text. The **Gr.text.width** command tells you width of the space in which a specific string will be drawn. The **Gr.get.textBounds** command tells you the locations of the left-, top-, right-, and bottom-most pixels actually drawn for a specific text string.

The **Gr.modify** parameters for **Gr.text.draw** are "x", "y", and "text". The value for "text" is a string representing the new text.

29.12.5 Gr.text.height

Syntax: Gr.text.height {<height_nvar>} {, <up_nvar>} {, <down_nvar>}

Returns height information for the current font and text size. All of the parameters are optional; use commas to indicate omitted parameters.

If the **height** variable <height_nvar> is present, it is set to the height in pixels of the space that will be used to print most text in most languages. This is the value you set with **Gr.text.size**. In typeface terminology, it is the "ascent" plus the "descent". The space contains the ascenders of letters like "h" and the descenders of letters like "y".

Some letters, such as the Polish letter "Ż", may not fit in this space. The position of a line high enough to contain all possible characters is returned in the **up** variable <up_nvar>, if it is present.

If the **down** variable <down_nvar> is present, it is set to the "descent" value. This position is low enough to contain the lowest part of all possible characters, such as the tail of a "y".

Gr.text.draw positions text so the the body of the characters sit on a line called the baseline. The **down** and **up** values are reported as offsets from this baseline. The **up** value is negative, because it defines a position above the baseline. The **down** value is positive, because its position is below the

baseline. The **height** value is not an offset, so it is always positive. **down** - **up** is always larger than **height**.

Sometimes you want to know the real screen positions of the top and bottom of the area where your text will be drawn, independent of the actual text you will draw there. The bottom of this area is the **y** coordinate of **Gr.text.draw** plus the **down** value of **Gr.text.height**. For most applications, the top of the text area is the bottom position minus the **height** value of **Gr.text.height**, so **y** + **down** - **height**. For some applications (such as a Polish text field), you may need the extra height you get with **y** + **up**.

```
Gr.text.size 40
Gr.text.height ht, up, dn              % ht is 40
Gr.text.draw t, x, y, "Hello, World!"
txtBottom = y + dn
txtTop = txtBottom - ht                % good for most applications
txtTop = y + up                        % high enough for all possible text
                                       % (up is negative)
```

29.12.6 Gr.text.setFont

Syntax: Gr.text.setFont {{<font_ptr_nexp>|<font_family_sexp>} {, <style_sexp>} {,<paint_nexp>}}

Set the text font, specifying typeface and style. Both of these parameters are optional. This command is similar to the older **Gr.text.typeFace**, but it is more flexible.

If the font parameter is a numerical expression <font_ptr_nexp>, it must be a font pointer value returned by the **Font.load** command. You cannot modify the style of a font once it is loaded, so the style parameter <style_sexp> is ignored.

If the font parameter is a string expression <font_family_sexp>, it must specify one of the font families available on your Android device. If your device does not recognize the string, the font is set to the system default font typeface. On most systems, the default is **"sans serif"**.

The standard font families are **"monospace"**, **"serif"**, and **"sans serif"**. Some more recent versions of Android also support **"sans-serif"**, **"sans-serif-light"**, **"sans-serif-condensed"**, and **"sans-serif-thin"**. The font family names are not case-sensitive: "Serif" or "SERIF" works as well as "serif".

If you omit the font parameter, the command sets the most recently loaded font (see **Font.load**). If you have deleted fonts (see **Font.delete**), the command sets the most recently loaded font that has not been deleted. If you have not loaded any fonts, or if you have cleared them (see **Font.clear**), the command sets the default font family.

If you specify a font family, you can use the style parameter <style_sexp> to change the font's appearance. The parameter value must be one of the style strings shown in the table below. You may use either the full style name or an abbreviation as shown. The parameter is not case sensitive: "BOLD", "bold", "Bold", and "bOlD" are all the same. If you use any other string, or if you omit the style parameter, the style is set to **"NORMAL"**.

Style Name	Abbreviation
"NORMAL"	"N"
"BOLD"	"B"
"ITALIC"	"I"
"BOLD_ITALIC"	"BI"

Notes: The **"monospace"** font family always displays as **"normal"**, regardless of the style parameter. Some devices do not support all of the styles.

You may use the optional Paint pointer parameter <paint_nexp> to specify a Paint object to modify. Normally this parameter is omitted. See **Gr.color**, *Advanced usage* for more information.

29.12.7 Gr.text.size

Syntax: Gr.text.size {{<size_nexp>}{,<paint_nexp>}}

Specifies the size of the text in pixels. The size <nexp> sets the nominal height of the characters. This height is large enough to include the top of characters with ascenders, like "h", and the bottom of characters with descenders, like "y". Character width is scaled proportionately to the height.

You may use the optional Paint pointer parameter <paint_nexp> to specify a Paint object to modify. Normally this parameter is omitted. See **Gr.color**, *Advanced usage* for more information.

29.12.8 Gr.text.skew

Syntax: Gr.text.skew {{<skew_nexp>}{,<paint_nexp>}}

Skews the text to give an italic effect. Negative values of <nexp> skew the bottom of the text left. This makes the text lean forward. Positive values do the opposite. Traditional italics can be best imitated with <nexp> = -0.25.

You may use the optional Paint pointer parameter <paint_nexp> to specify a Paint object to modify. Normally this parameter is omitted. See **Gr.color**, *Advanced usage* for more information.

29.12.9 Gr.text.strike

Syntax: Gr.text.strike {{<lexp>}{,<paint_nexp>}}

Turns overstrike on or off on text objects drawn after this command is issued:

- If the value of the strike parameter <lexp> is false (0), text strike is turned off.
- If the parameter value is true (not zero), text will be drawn with a strike-through line.
- If the parameter is omitted, the bold setting is toggled.

You may use the optional Paint pointer parameter <paint_nexp> to specify a Paint object to modify. Normally this parameter is omitted. See **Gr.color**, *Advanced usage* for more information.

29.12.10 Gr.text.typeFace

Syntax: Gr.text.typeFace {{<font_nexp>} {, <style_nexp>} {,<paint_nexp>}}

Set the text typeface and style. Both of these parameters are optional. The default value if you omit either parameter is 1. All valid values and their meanings are shown in this table:

The values for <font_nexp> are:		The values for <style_nexp> are:	
1	Default font	1	Normal (not bold or italic)
2	Monospace font	2	Bold
3	Sans-serif font	3	Italic
4	Serif font	4	Bold and italic

This command is similar to the newer **Gr.text.setFont**, except that it is limited to the four typefaces listed in the table. It cannot specify fonts loaded by the **Font.load** command.

Notes: The "Monospace" font (font 2) always displays as "Normal" (style 1), regardless of the style parameter. Some devices do not support all of the styles.

You may use the optional Paint pointer parameter <paint_nexp> to specify a Paint object to modify.

Normally this parameter is omitted. See **Gr.color**, *Advanced usage* for more information.

29.12.11 Gr.text.underline

Syntax: Gr.text.underline {{<lexp>}{,<paint_nexp>}}

Turns underlining on or off on text objects drawn after this command is issued:
- If the value of the underline parameter <lexp> is false (0), text underlining is turned off.
- If the parameter value is true (not zero), drawn text will be underlined.
- If the parameter is omitted, the underline setting is toggled.

You may use the optional Paint pointer parameter <paint_nexp> to specify a Paint object to modify. Normally this parameter is omitted. See **Gr.color**, *Advanced usage* for more information.

29.12.12 Gr.text.width

Syntax: Gr.text.width <nvar>, <exp>

Returns the pixel width of a string (from <exp>) in the variable <nvar>.

- If the parameter <exp> is a string expression, the return value is the width of the string as if it were displayed on the screen using the latest text attribute settings: the typeface, size, and style as set by the **Gr.text.*** commands (or default values if you did not set them).
- If the parameter <exp> is a numeric expression, its value must be a text object number from **Gr.text.draw**, or you will get a run-time error. The return value is the width of the text of the object as it would be displayed on the screen by **Gr.render**.

Advanced usage: To calculate dimensions, both **Gr.text.width** and **Gr.get.textBounds** (below) use a text string and a set of text attributes. The text attributes are kept in a Paint object. The source of the string and the Paint depends on the type of the <exp> parameter:

If <exp> is a	value of <exp> is	source of text string is	Source of text attributes is
string expression	the string to measure	value of <exp>	Current Paint (most recent **Gr.text.*** settings)
numeric expression	a text object number from **Gr.text.draw**	string from **Gr.text.draw** (kept in text object)	Paint attached to the text object (see **Note**, below)

Note: **Gr.text.draw** attaches a Paint to the text object using the text attributes that are current at that time. This is the Paint **Gr.render** uses to display the text on the screen. If you modify this Paint, the changes are reflected in values returned by **Gr.text.width** and **Gr.get.textBounds**, and also shown on the screen with the next **Gr.render.**

29.12.13 Gr.text.wrap

Syntax: Gr.text.wrap ResultArray$[], <text_sexp>, <widthPx_nexp> {{, <endChecks_sexp>}, <paint_nexp>}

Returns <text_sexp> divided into array of strings ResultArray$[], so that each string fits into <widthPx_nexp> pixels. The <endChecks_sexp> specifies the characters where a line break is possible. If it is an empty string, the breaks can be in the middle of a word. The default is "!,;. ". If <paint_nexp> is not specified, the current paint settings are used.

Example:

```
Gr.open "_White", 1000, -1
Gr.drawable.load gdo, "fly.gif"
Gr.drawable.draw go, gdo, 100,100,900,900
Gr.drawable.start go
```

```
text$ = "Hello Basic! users!\n"
text$ += "This is a text created for testing the Gr.text.wrap command.\n"
text$ += "We all hope, you have fun with all the flavors of BASIC!"
widthPx = 800

Gr.color "_Black"
Gr.text.size 100

Gr.text.bold 1
Bundle.put myPaintBundle, "_TextSize", 65
Bundle.put myPaintBundle, "_TextScaleX", 1.7
Bundle.put myPaintBundle, "_Color", "_Red"
Gr.paint.set myPaintBundle                % Overwrites the current Paint
Gr.text.wrap ResultArray$[], text$, widthPx
Array.length al, ResultArray$[]
Bundle.get myPaintBundle, "_TextSize", lineSpace
lineSpace *= 1.1     % Same as lineSpace = LineSpace * 1.1

For i = 1 To al
 Gr.text.draw dp, 100, 100 + i * lineSpace, ResultArray$[i]
Next
Gr.color "_Gray", 0
Gr.rect rp, 100,100, 100 + widthPx, 100 + al * lineSpace + lineSpace / 1.1 * 0.3

Gr.render
```

29.13 Graphics Touch Commands

If the user touches the screen and then moves the finger without lifting the finger from the screen, the motion can be tracked by repeatedly calling on the touch query commands. This will allow you to program the dragging of graphical objects around the screen. The Sample Program, f23_breakout.bas, illustrates this with the code that moves the paddle.

The **OnGrTouch:** label can be used optionally to interrupt your program when a new touch is detected.

The touch commands report on one- or two-finger touches on the screen. If the two fingers cross each other on the x-axis then touch and touch2 will swap.

29.13.1 Gr.array.touch

Syntax: Gr.array.touch Array[], <count_nvar>

Returns all available touched points as an array of pairs (x, y), independent if they are touched or moved. The variable <count_nvar> returns the number of points. This command should be placed between **OnTimer:** and **Timer.resume**, for best stability. **Set.timer** is recommended with 10 or more.

29.13.2 Gr.bounded.touch

Syntax: Gr.bounded.touch touched, left, top, right, bottom

The Touched parameter will be returned true (not zero) if the user has touched the screen within the rectangle defined by the left, top, right, bottom parameters. If the screen has not been touched or has been touched outside of the bounding rectangle, the touched parameter will be return as false (zero). The command will continue to return true as long as the screen remains touched and the touch is within the bounding rectangle.

The bounding rectangle parameters are for the actual screen size. If you have scaled the screen then you need to similarly scale the bounding rectangle parameters. If the parameters that you used in **Gr.scale** were scale_x and scale_y (**Gr.scale scale_x, scale_y**) then divide left and right by scale_x and divide top and bottom by scale_y.

29.13.3 Gr.bounded.touch2

Syntax: Gr.bounded.touch2 touched, left, top, right, bottom

Same as **Gr.bounded.touch** except that it reports on second simultaneous touch of the screen.

WARNING: Use this command with caution, as the event handler works like a tennis player batting against a much too fast ball-machine. If possible, use **Gr.array.touch** or **Gr.list.touch**, as these commands recognize the x-y status of one or more fingers at the **same** time.

29.13.4 Gr.last.touch

Syntax: Gr.last.touch <last_index_nvar>, <x_nvar>, <y_nvar>

Returns the last touched index in <last_index_nvar> and the coordinates of the touch in <x_nvar> and <y_nvar>. If the screen is not currently touched, <last_index_nvar> returns 0, and <x_nvar> and <y_nvar> return the coordinates of the last touch. If the screen has never been touched, <x_nvar> and <y_nvar> are left unchanged.

The returned values are relative to the actual screen size. Thus if you scaled the screen, you need to scale the returned parameters in the opposite direction.

29.13.5 Gr.list.touch

Syntax: Gr.list.touch <listX(Y)_pointer_nexp>, {<listY_pointer_nexp>}, <count_nvar>

Returns all available touched points, independent if these are touched or moved, as one list of pairs (x, y) if <listY_pointer_nexp> is not defined. When <listY_pointer_nexp> is used, it's list contains the Y value of the point and the list <listX(Y)_pointer_nexp> the X value. The variable <count_nvar> returns the number of points. This command should be placed behind **onTimer:** and before **Timer.resume**, for best stability. **Set.timer** is recommended with 10 or more.

29.13.6 Gr.onGrTouch.resume

Syntax: Gr.onGrTouch.resume

This statement should be placed at the end of the interrupt handler at **OnGrTouch:**.

29.13.7 Gr.onGrTouchMove.resume

Syntax: Gr.onGrTouchMove.resume

This statement should be placed at the end of the interrupt handler at **OnGrTouchMove:**.

29.13.8 Gr.onGrTouchUp.resume

Syntax: Gr.onGrTouchUp.resume

This statement should be placed at the end of the interrupt handler at **OnGrTouchUp:.**

Example:

```
Fn.def swipeDirection(bx, by, ex, ey, t1, t2, vMinQ)
 sQ = (ey - by) ^ 2 + (ex - bx) ^ 2
 t = t2 - t1
 vQ = sQ / t
 mDirection = Atan2(ey - by, ex - bx)
 mDirection = Round(mDirection / Pi() * 2)
 If mDirection = -1 Then mDirection = 3
 If vQ > vMinQ Then
  Fn.rtn Abs(mDirection)
 Else
```

```
    Fn.rtn -1
  EndIf
Fn.end

! 200 from 255 alpha; 1000 no status bar and no navigation bar
Gr.open "_200,DarkBlue", 1000, -1
Gr.screen osx, osy, density, isR, lo[]
! It is strongly recommended, to use the <layout[ ]> array
sx = lo[3] - lo[1] : sy = lo[4] - lo[2] %[Left, Top, Right, Bottom].
! Device independent; v², because we save the square root later
vMinQ = (4 * density / 160) ^ 2
Gr.color "_Red", 1
Gr.circle goc, sx / 2, sy / 2, sx / 50
Gr.set.stroke 5
Gr.text.size sy / 30
mDir$ = "Direction = "
Gr.text.draw vd, 20, sy / 18, mDir$

Gr.render
mPause = 150
Do
 Pause mPause
Until 0

OnGrTouch:
 mPause = 20
 mSwipe = 1 : t1 = Clock()
 Gr.touch touched, mSwipeXb, mSwipeYb
 ! Print "TOUCH"
Gr.OnGrTouch.resume

OnGrTouchMove:
 mSwipe++
 Gr.touch touched, cx, cy
 Gr.modify goc, "x", cx, "y", cy
 Gr.render
 ! Print "MOVE"
Gr.OnGrTouchMove.resume

OnGrTouchUp:
 If mSwipe > 2 Then
  Gr.Touch touched, mSwipeXe, mSwipeYe
  mDirecetion = swipeDirection(mSwipeXb, mSwipeYb, mSwipeXe, mSwipeYe, t1, ~
   Clock(), vMinQ)
 ! Print "SWIPE: "; mDirecetion
 Gr.modify vd,"text", mDir$ + Int$(mDirecetion)
 Gr.render
 EndIf
 mPause = 100
 ! Print "UP"
Gr.OnGrTouchUp.resume
```

29.13.9 Gr.scale.touch

Syntax: Gr.scale.touch x_factor, y_factor{{{, x_distance}, y_distance}, <reverse_nexp>}

Scale all touch commands by x and y scale factors, and optionally translate by x and y distance. This command is provided to allow you to touch in a device-independent manner, scaling and translating the touches to the actual size of the screen that your program is running on.

If <reverse_nexp> set to 1, you can insert the same values used in **Gr.scale** to compensate its scale and translation values. Default is 0, in which case you have to compute the values yourself.

29.13.10 Gr.touch

Syntax: Gr.touch touched, x, y

Tests for a touch on the graphics screen. If the screen is being touched, Touched is returned as true (not 0) with the (x,y) coordinates of the touch. If the screen is not currently touched, Touched returns false (0) with the (x,y) coordinates of the last previous touch. If the screen has never been touched, the x and y variables are left unchanged. The command continues to return true as long as the screen remains touched.

If you want to detect a single short tap, after detecting the touch, you should loop until touched is false.

```
Do
 Gr.touch touched, x, y
Until touched

! Touch detected, now wait for finger lifted
Do
 Gr.touch touched, x, y
Until !touched
```

The returned values are relative to the actual screen size. If you have scaled the screen then you need to similarly scale the returned parameters. If the parameters that you used in **Gr.scale** were scale_x and scale_y (**Gr.scale scale_x, scale_y**) then divide the returned x and y by those same values.

```
Gr.touch touched, x, y
Xscaled = x / scale_x
Yscaled = y / scale_y
```

29.13.11 Gr.touch2

Syntax: Gr.touch2 touched, x, y

The same as **Gr.touch** except that it reports on second simultaneous touch of the screen.

WARNING: Use this command with caution, as the event handler works like a tennis player batting against a much too fast ball-machine. If possible, use **Gr.array.touch** or **Gr.list.touch**, as these commands recognize the x-y status of one or more fingers at the **same** time.

29.13.12 OnGrTouch:

Syntax: OnGrTouch:

Label for an interrupt handler which traps any touch on the Graphics screen. When done, execute the **Gr.onGrTouch.resume** statement to resume the interrupted program.

To detect touches on the Output Console (not in Graphics mode), use **OnConsoleTouch:**.

To detect a double tap use:

```
grTapCounter = 0
Print "GR Double Tap Example"
Do
Until 0
End "Bye...!"

OnTimer:
 If grTapCounter = 1 Then
  Print "Only One Tap!"
  Timer.clear
  grTapCounter = 0
 EndIf
Timer.resume

OnGrTouch:
 grTapCounter++
 If grTapCounter = 1 Then
  Timer.clear
```

```
    Timer.set 500
  EndIf
  If grTapCounter = 2 Then
   Print "Double Tap!"
   Timer.clear       %Needed, if you will not end execution.
   grTapCounter = 0  %Needed, if you will not end execution.
   Pause 2000
   End "Bye...!"
  EndIf
 Gr.onGrTouch.resume
```

Note: The **Sched.set** command can work as a timer, too.

29.13.13 OnGrTouchMove:

Syntax: OnGrTouchMove:

Label for an interrupt handler which traps any touch **move** on the Graphics screen. When done, execute the **Gr.onGrTouchMove.resume** statement to resume the interrupted program.

29.13.14 OnGrTouchUp:

Syntax: onGrTouchUp:

Label for an interrupt handler which traps any touch **up** on the Graphics screen. When done, execute the **Gr.onGrTouchUp.resume** statement to resume the interrupted program.

29.14 Graphics Visibility Commands

29.14.1 Gr.behind

Syntax: Gr.behind <object_behind_nexp>, <object_number_nexp>

Moves the object with the specified object number <object_behind_nexp> behind the object <object_number_nexp> visually. That means the element <object_behind_nexp> is in front of the one specified by <object_number_nexp> on the display list. This change will not be visible until the **Gr.render** command is called. Note, this command works only in conjunction with **shown single** objects.

29.14.2 Gr.hide

Syntax: Gr.hide <object_number_nexp>{, <object_number_nexp> ...}

Hides the object with the specified Object Number. If the Object is a Group, all of the Graphical Objects in the Group are hidden. This change will not be visible until the **Gr.render** command is called.

29.14.3 Gr.inFront

Syntax: Gr.inFront <object_inFront_nexp>, <of_object_nexp>

Moves the object with the specified object number <object_inFront_nexp> in front of the object <of_object_nexp> visually. That means the element <object_inFront_nexp> is behind the one specified by <of_object_nexp> on the display list. This change will not be visible until the **Gr.render** command is called. Note, this command works only in conjunction with **shown single** objects.

29.14.4 Gr.show

Syntax: Gr.show <object_number_nexp>{, <object_number_nexp> ...}

Shows (unhides) the object with the specified Object Number. If the Object is a Group, all of the Graphical Objects in the Group are shown. This change will not be visible until the **Gr.render** command is called.

29.14.5 Gr.show.toggle

Syntax: Gr.show.toggle <object_number_nexp>

Toggles visibility of the object with the specified Object Number. If it is hidden, it will be shown. If it is shown, it will be hidden. If the Object is a Group, all of the Graphical Objects in the Group are toggled. This change will not be visible until the **Gr.render** command is called.

29.14.6 Gr.toBack

Syntax: Gr.toBack <object_number_nexp>{, <object_number_nexp> ...}

Moves the object with the specified object number visually to the back. That means this element is the first one on the display list. If more than one object is specified, this command works in the order of the specified objects. This change will not be visible until the **Gr.render** command is called. Note, this command works only in conjunction with **shown single** objects.

29.14.7 Gr.toFront

Syntax: Gr.toFront <object_number_nexp>{, <object_number_nexp> ...}

Moves the object with the specified object number visually to the front. That means this element is the last one on the Display List. If more than one object is specified, this command works in the order of the specified objects. This change will not be visible until the **Gr.render** command is called. Note, this command works only in conjunction with **shown single** objects.

29.15 Graphics Miscellaneous Commands and Functions

29.15.1 Color

Syntax: nvar = Color(<sexp>)

Returns the color number of a color string expression.

{Alpha,}Red,Green,Blue (comma delimited string) or _{Alpha,}ColorName ({comma delim.} string) or #{hn}hnhnhn (hex. string)	Default is "255,0,0,0" or "_Black", "_HSV240" or "#ff000000"

See also: **Gr.color**

29.15.2 Color$

Syntax: svar = Color$(<nexp>)

Returns a color string expression of a system color number of the type "0-255, 0-255, 0-255, 0-255" (alpha, red, green, blue).

29.15.3 Gr.clip

Syntax: Gr.clip <object_ptr_nexp>, <left_nexp>, <top_nexp>, <right_nexp>, <bottom_nexp>{, <RO_nexp>}

Objects that are drawn after this command is issued will be drawn only within the bounds (clipped) of the clip rectangle specified by the "left, top, right, bottom" numeric expressions.

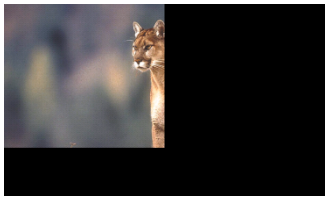
The final parameter is the Region Operator, <RO_nexp>. The Region Operator prescribes how this clip will interact with everything else you are drawing on the screen or bitmap. If you issue more than one **Gr.clip** command, the RO prescribes the interaction between the current **Gr.clip** rectangle and the previous one. The RO values are:

0	**Intersect**
1	**Difference**

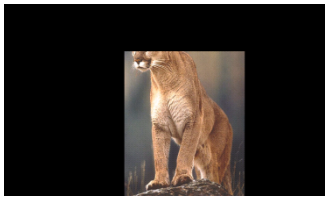
The Region Operator parameter is optional. If it is omitted, the default action is **Intersect**.

Examples:

Original

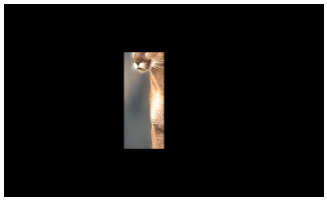


Clip 1

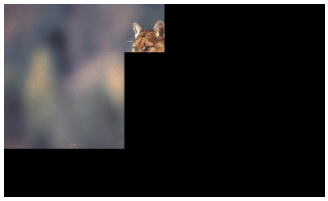


Clip 2

Clip 2 applied to Clip 1 with RO parameter on Clip 2



0 = Intersect



1 = Difference

Gr.clip is a display list object. It can be modified with **Gr.modify**. The modify parameters are "left", "top", "right", "bottom", and "RO".

The **Gr.show** and **Gr.hide** commands can be used with the **Gr.clip** object.

RO values other than INTERSECT and DIFFERENCE have the ability to expand the clip, so values from 2 and up are ignored. The graphic mode clipping APIs are intended to only expand the clip as a result of a restore operation. This enables a view parent to clip a canvas to clearly define the maximal drawing area of its children. The recommended alternative calls are **Gr.clip** with <RO_nexp> less than 2 and **Gr.clipOut**.

See also: **Gr.clipOut**, **Gr.color** with its optional xFermode.

29.15.4 Gr.clipOut

Syntax: Gr.clipOut <object_ptr_nexp>, <left_nexp>, <top_nexp>, <right_nexp>, <bottom_nexp>

Sets the clip to the difference of the current clip and the specified rectangle, which is expressed by the "left, top, right, bottom" numeric expressions.

29.15.5 Gr.get.bounds

Gr.get.bounds <object_ptr_nexp>, <left_nvar>, <top_nvar>, <right_nvar>, <bottom_nvar>

Gets the boundary rectangle of a graphic object as it would be drawn on the screen. The returned coordinate values give you the dimensions of the bounding rectangle but not its location.

Note that objects turned by **Gr.rotate.start / Gr.rotate.end** do not return their new location and angle.

If an object type is not supported, all variables return 0.

Supported object types are: arc, bitmap, circle, drawable, line, oval, point, rect and text.

29.15.6 Gr.getDL

Syntax: Gr.getDL <dl_array[]> {, <keep_all_objects_lexp> }

Writes the current Display List into the numeric array <dl_array[]>. The array is specified without an index. If the array exists, it is overwritten. Otherwise a new array is created. The result is always a one-dimensional array. If the Display List is empty, the array will have one entry that does not display anything.

By default, objects hidden with **Gr.hide** are not included in the returned array. To get all objects, including hidden objects, set the optional keep_all_objects flag to true (any non-zero value).

29.15.7 Gr.get.params

Syntax: Gr.get.params <object_ptr_nexp>, <param_array$[]>

Get the modifiable parameters of the specified display list object. The parameter strings are returned in the <param_array$[]> in no particular order. The array is specified without an index. If the array exists, it is overwritten. Otherwise a new array is created. The result is always a one-dimensional array.

For a complete list of parameters, see the table in **Gr.modify**.

29.15.8 Gr.get.pixel

Syntax: Gr.get.pixel x, y, alpha, red, green, blue

Returns the color data for the screen pixel at the specified x, y coordinate. The x and y values must not exceed the width and height of the screen and must not be less than zero.

To get a pixel from the screen, BASIC! must first create a bitmap from the screen. If there is not enough memory available to create the bitmap, you will get an "out-of-memory" run-time error.

29.15.9 Gr.get.position

Syntax: Gr.get.position <object_ptr_nexp>, x, y

Get the current x, y position of the specified display list object. If the object was specified with rectangle parameters (left, top, right, bottom) then left is returned in x and top is returned in y. For Line objects, the x1 and y1 parameters are returned.

29.15.10 Gr.get.type

Syntax: Gr.get.type <object_ptr_nexp>, <type_svar>

Get the type of the specified display list object. The type is a string that matches the name of the command that created the object: "point", "circle", "rect", etc. For a complete list of types, see the table in **Gr.modify**.

If the <object_ptr_nexp> parameter does not specify a valid display list object, the returned type is the empty string, "". You can call the **GetError$()** function to get information about the error.

29.15.11 Gr.get.value

Syntax: Gr.get.value <object_ptr_nexp> {, <tag_sexp>, <value_nvar | value_svar>}...

The value of the parameter named <tag_sexp> ("left", "radius", etc.) in the Display List object <object_ptr_nvar> is returned in the variable <value_nvar> or <value_svar>. This command can return values from only one object at a time, but you may list as many tag/variable pairs as you want.

Most parameters are numeric. Only the **Gr.text.draw** "text" parameter is returned in a string var. The parameters for each object are given with descriptions of the commands in this manual. For a complete list of parameters, see the table in **Gr.modify**.

29.15.12 Gr.modify

Syntax: Gr.modify <object_ptr_nexp> {, <tag_sexp>, <value_nexp | value_sexp>}...

The value of the parameter named <tag_sexp> in the Display List object <object_ptr_nvar> is changed to the value of the expression <value_nexp> or <value_sexp>. This command can change only one object at a time, but you may list as many tag/value pairs as you want.

With this command, you can change any of the parameters of any object in the Display List. The parameters you can change are given with the descriptions of the commands in this manual. In addition there are two general purpose parameters, "paint" and "alpha" (see below for details). You must provide parameter names that are valid for the specified object.

The results of **Gr.modify** commands will not be observed until a **Gr.render** command executes.

The following table shows the possible parameters:

	TYPE	POSITION 1 (numeric)		POSITION 2 (numeric)		ANGLE/ RADIUS (numeric)	UNIQUE (various)	PAINT (list ptr)	ALPHA (num)
SHAPES and OBJECTS	arc	left	top	right	bottom	start_angle sweep_angle	fill_mode	paint	alpha
	arcpoly	x	y				list	paint	alpha
	bitmap	x	y				bitmap	paint	alpha
	circle	x	y			radius		paint	alpha
	drawable	left	top	right	bottom		drawable	paint	alpha
	line	x1	y1	x2	y2			paint	alpha
	oval	left	top	right	bottom			paint	alpha
	pixels	x	y					paint	alpha
	point	x	y					paint	alpha
	poly	x	y				list	paint	alpha
	rect	left	top	right	bottom	rx, ry		paint	alpha
	text	x	y				text	paint	alpha
MODI-FIERS	clip	left	top	right	bottom		RO	paint	alpha
	clipout	left	top	right	bottom			paint	alpha
	group						list	paint	alpha
	rotate	x	y			angle		paint	alpha

TABLE NOTES:

- The **TYPE** column shows the string returned by **Gr.get.type** for each graphical object type.
- **Gr.get.position** returns the values in the **POSITION 1** columns.
- All table entries are **Gr.modify** tags (strings). Values of all the tags are numeric except for **"text"**.
- The values of tags in the **UNIQUE** column are either strings (**"text"**) or numbers with special interpretations. **"fill_mode"** is a logical value. **"list"** is a pointer to a list of point coordinates. **"RO"** is a Region Operator as explained in **Gr.clip**.
- **"alpha"** is an integer value from 0 to 256, with 256 interpreted specially. See **General Purpose Parameters**, below.
- You can modify the **Gr.set.pixels** point-coordinates array directly. There is no **Gr.modify** tag.

For example, suppose a bitmap object was created with:

```
Gr.bitmap.draw BM_ptr, galaxy_ptr, 400, 120
```

Executing `gr.modify BM_ptr, "x", 420` would move the bitmap from x =400 to x = 420.
Executing `gr.modify BM_ptr, "y", 200` would move the bitmap from y = 120 to y = 200.
Executing `gr.modify BM_ptr, "x", 420, "y", 200` would change both x and y at the same time.
Executing `gr.modify BM_ptr, "bitmap", Saturn_ptr` would change the bitmap of an image of a (preloaded) Galaxy to the image of a (preloaded) Saturn.

General Purpose Parameters

When you create a graphical object, all the graphics settings (color, stroke, text settings, and so forth) are captured in a Paint object. You can use the "paint" parameter to replace the Paint object, changing any graphics setting you want to. See the **Gr.paint.get** command description (below) for more details.

Normally, graphical objects get their alpha channel value (transparency) from the latest **Gr.color** command. You can change the "alpha" parameter to any value from 0 to 255. Setting alpha to 256 tells BASIC! to use the alpha from the latest color value.

For example, you can make an object slowly appear and disappear, just by changing its alpha with **Gr.modify**.

```
Do
  For a = 1 To 255 Step 10
    gr.modify object,"alpha",a
    gr.render
    pause 250
  Next a

  For a = 255 To 1 Step -10
    gr.modify object,"alpha",a
    gr.render
    pause 250
  Next a
Until 0
```

29.15.13 Gr.move

Syntax: Gr.move <object_ptr_nexp> {{, dx}{, dy}}

Moves the graphics object by the amounts dx and dy. If the object is a group, all of the graphical objects in the group are moved. The dx and dy parameters are optional. If omitted they default to 0.

29.15.14 Gr.newDL

Syntax: Gr.newDL <dl_array[{<start>,<length>}]>

Replaces the existing display list with a new display list read from a numeric array (dl_array[]) or array segment (dl_array[start,length]) of object numbers. Zero values in the array will be treated as null

objects in the display list. Null objects will not be drawn nor will they cause run-time errors.

See the Display List subtopic in this chapter for a complete explanation.

See the Sample Program file, f24_newdl, for a working example of this command.

29.15.15 GR.onGrScreen.resume

Syntax: GR.onGrScreen.resume

This statement should be placed at the end of the interrupt handler at **OnGrScreen:**.

29.15.16 Gr.save

Syntax: Gr.save <filename_sexp> {,<quality_nexp>}

Saves the current screen to a file. The default path is "<pref base drive>/rfo-basic/data/".

The file will be saved as a JPEG file if the filename ends in ".jpg".

The file will be saved as a PNG file if the filename ends in anything else (including ".png").

The optional <quality_nexp> is used to specify the quality of a saved JPEG file. The value may range from 0 (bad) to 100 (very good). The default value is 50. The quality parameter has no effect on PNG files which are always saved at the highest quality level.

Note: The size of the JPEG file depends on the quality. Lower quality values produce smaller files.

29.15.17 Gr.screen.to_bitmap

Syntax: Gr.screen.to_bitmap <bm_ptr_nvar>

The current contents of the screen will be placed into a bitmap. The pointer to the bitmap will be returned in the bm_ptr variable. If there is not enough memory available to create the bitmap, the returned bitmap pointer is -1. Call **GetError$()** for information about the failure.

Please note the idiosyncratic underscore in the command.

If a camera preview is behind the graphic screen, the preview is not visible. So take a photo, convert (scale, crop) its size fitted to the screen, put the result in the graphic screen background, start **Gr.screen.to_bitmap** and delete the background.

29.15.18 Gr.target.modify

Syntax: Gr.target.modify <target_sexp>, object_ptr_Array[], inp_1_Array[] {{, inp_2_Array[] }, inp_3_Array[], inp_4_Array[]}

Modifies graphic elements specified by the object_ptr_Array[]. The desired target is given by <target_sexp>.

Target	Element Types	inp_1_Array[]	inp_2_Array[]	inp_3_Array[]	inp_4_Array[]
_Position	Like Position 1 All**(!)** element types, but without ***group***	X	X		
_Frame	Like Position 1 + 2 All element types with four parameters	X	X	X	X
_Move	Like Position 1	X	X		

Target	Element Types	inp_1_Array[]	inp_2_Array[]	inp_3_Array[]	inp_4_Array[]
	But moves the elements relative All**(!)** element types				
_Start_Sweep_ Angles	Element type ***arc***	X			
_Radius	Element type ***circle***	X			
_Corner_Radii	Element type ***rect***	X	X		
_Angle	Element type ***rotate***	X			
_Fill_Mode	Element type ***arc***	X			
_Bitmap	Element type ***bitmap***	X			
_Drawable	Element type ***drawable***	X			
_List	Element types ***arcpoly, path, poly, group***	X			
_RO	Element type ***RO***	X			
_Paint	Like Paint All element types	X			
_Alpha	Like Alpha All element types	X			

The results of **Gr.target.modify** commands will not be observed until a **Gr.render** command executes.

See also: **Gr.modify, Gr.move**

Example:

```
Gr.target.modify "_Drawable", dAbles[], dAbleLeft[], dAbleTop[], dAbleRight[],
dAbleBottom[]
```

29.15.19 Gr_collision

Syntax: Gr_collision(<object_1_nexp>, <object_2_nexp>{,<dist_1_nvar>, <dist_2_nvar>})

Gr_collision() is a function, not a command. The variables <object_1_nvar> and <object_2_nvar> are the object pointers returned when the objects were created.

If the boundary boxes of the two objects overlap then the function will return true (not zero). If they do not overlap then the function will return false (zero).

Objects that may be tested for collision are: rectangle, bitmap, circle, arc, oval, and text. In the case of a circle, an arc, an oval, or text, the object's rectangular boundary box is used for collision testing, not the actual drawn object.

Objects that may be tested for collision are: arc, bitmap, circle, drawable, line, oval, point, rect and text. In case of a circle, an arc, a line, an oval, or text, the object's rectangular boundary box is used for collision testing, not the actual drawn object.

Note that objects turned by **Gr.rotate.start** / **Gr.rotate.end** do not return its new location and angle. In this case he collision testing will fail.

The arguments <dist_1_nvar> and <dist_2_nvar> allows to set back (-) or forth (+) the collision border. In a case of a backset, make sure that your object is big enough.

Example:

```
% A good example is a circle that touches a rectangle.
% The circle has a diameter of 50 so a radius of 25.
% With a circle backset of -25 (now a point) and a rectangle
% forthset of +25 the circle touches exactly the rectangle.

Gr_collision(circle50, rectangle, -25, +25)
```

29.15.20 List.target.modify

Syntax: List.target.modify <list_ptr_nexp>, <target_sexp>, subObject_ptr_Array[], inp_1_Array[] {, inp_2_Array[], <points_nexp>}

Modifies graphic elements specified by the <subObject_ptr_Array[]>. The desired target is given by <target_sexp>.

The optional <inp_2_Array[]> and <points_nexp> are **only** used in conjunction with the **Gr.poly** enhancement to use more than one polygon with the same number of points by one command and one graphic object. In this case, <object_ptr_Array[]> specifies the polygon object in this sub list.

If <inp_2_Array[]> and <points_nexp> are specified the targets **_Position** and **_Move** can be changed.

If only <inp_1_Array[]> is specified the targets **_Paint** and **_Alpha** can be changed.

Command syntax of **Gr.poly**:

```
Gr.poly <obj_nvar>, <list_pointer_nexp> {{{{, x, y}, <closed_nexp>},
<pointsPerPoly_nexp>}, <paintPointerList_nexp>}
```

Target	Element Types	inp_1_Array[]	inp_2_Array[]
_Position	Like Position 1	X	X
_Move	Like Position 1 But moves the elements relative	X	X
_Paint	Like Paint	X	
_Alpha	Like Alpha	X	

The results of _Position and _Move have to be part of a **Gr.modify** command, but that will not be observed until a **Gr.render** command executes.

Note that the results of _Paint and _Alpha are mapped by each following **Gr.render** command **without** a previous **Gr.modify**.

See also: **Gr.modify**, **Gr.move**, **Gr.paint**

Example:

```
List.target.modify "_Move", listOfPoints, pointsOfPolygon, subObjectPtrArray[],
xArray[], yArray[]
```

29.15.21 OnGrScreen:

Syntax: OnGrScreen:

Label for an interrupt handler which traps on receive a screen layout change.

Example:

```
Fn.def SetViewSizes (globals)
 Gr.screen w, h, density, isRound , layout[ ]
 w = (layout[3] - layout[1])
 h = (layout[4] - layout[2])
 layoutRatio = w/h
 Bundle.put globals, "layoutW", w
 Bundle.put globals, "layoutH", h
 Fn.rtn layoutRatio
Fn.end

Fn.def DrawLandscape (globals)
 Bundle.get globals, "layoutW", w
 Bundle.get globals, "layoutH", h
 Print "Draw Landscape"
 Fn.rtn 1
FN.END

Fn.def DrawPortrait (globals)
 Bundle.get globals, "layoutW", w
 Bundle.get globals, "layoutH", h
 Print "Draw Portrait"
 Fn.rtn 1
Fn.end

Bundle.create globals
Print globals

Print "An OliBasic Solution"
Gr.open "_Blue", , -1 % Orientation changeable

Do
 Pause 20
Until 0

OnGrScreen:    % Interrupt will also trap GR.OPEN
 layoutRatio = SetViewSizes (globals)
 If layoutRatio > 1 Then
  DrawLandscape(globals)
 Else
  DrawPortrait(globals)
 EndIf
Gr.onGrScreen.resume
```

29.15.22 Within

Syntax: Within(<sexp>, <arg1>, <arg2>, <arg3>, <arg4>...)

This function returns <> 0 (true) if the search for a graphical object described by <sexp> is successful. In other cases 0.0 false is returned.

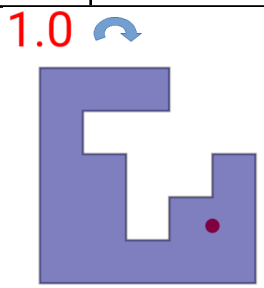
Example 1: if <sexp> is "_point_polygon" (read as "**With** point **in** polygon?"), then

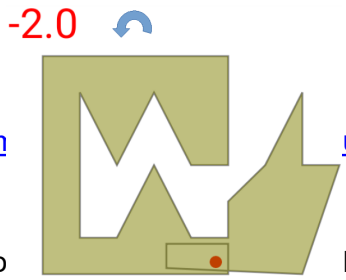
If the point is directly onto a line, it is true also.
Is a positive number returned, the polygon was drawn clockwise.
Is a negative number returned, the polygon was drawn counterclockwise.

If +1 or -1 is returned, the point is inside the polygon.
If +2 or 2 is returned, the point is inside an inner polygon of the polygon.

The following table describes the arguments for <sexp> = "_point_polygon".

Argument number	Description	Value type	Value
1	Point x	Double	Double
2	Point y	Double	Double
3	List of polygon points as x values or x, y pairs.	List numeric	Double
4	List of polygon points as y values (optional).	List numeric	Double

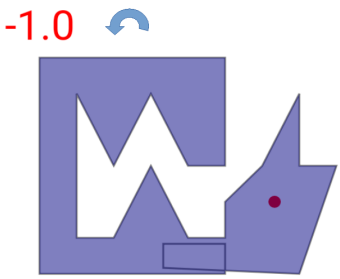




For more information see: http://geon usion.html.

Example 2: if <sexp> is "_polygon_p lygon **in** polygon?"), then if at least one point of the first polygon is inside or is directly onto a line of the second polygon, it returns true (1).

The following table describes the arguments for <sexp> = "_polygon_polygon".



30 HTML Commands

HTML commands allow you to interact with the Android Webview HTML engine control to create user interfaces using HTML and JavaScript. The interface provides for interaction between the HTML engine and your program. The HTML programmer can use JavaScript to send messages to your program. The HTML engine will also report user events such as the BACK key, hyperlink transfers, downloads, form data and errors.

The demo program, f37_html_demo.bas, combined with the HTML demo files, htmlDemo1.html and htmlDemo2.html, illustrate the various commands and possibilities. It is highly recommended that all three files be carefully studied to fully understand this interface.

Another demo program, f38_html_edit.bas, can be used to edit HTML files. To use the program, run it and enter the HTML file name without the html extension. The program will add the extension ".html".

30.1 Html.clear.cache

Syntax: Html.clear.cache

Clears the HTML cache.

30.2 Html.clear.history

Syntax: Html.clear.history

Clears the HTML history.

30.3 Html.close

Syntax: Html.close

Closes the HTML engine and display.

30.4 Html.evaluate.js

Syntax: Html.evaluate.js <js_sexp>

Sends a JavaScript script to the current HTML page or JavaScript if loaded. This command waits until the HTML page returns a result from the sender script. A complete function script can also be used without a loaded script. The result will be posted via **Html.get.datalink** with the header "EJS:".

An example of another way to call a function is

```
Html.load.URL "javascript:text('BASIC! for ever!')"
```

as seen in the sample program "data/htmldemo1.html".

Example:

```
closeHtml = 0
Html.open 0, -1
Html.load.string "<script>function callJS(str) { return str; } <\script>"
Html.evaluate.js "callJS( 'BASIC! for ever!' )"
Do
 Pause 100
Until closeHtml
Html.close
OnHtmlReturn:
 Html.get.datalink data$
 If Is_in("EJS:", data$) = 1 Then Print MID$(data$, 5)
```

```
 closeHtml = 1
Html.onHtmlReturn.resume
```

30.5 Html.get.datalink

Syntax: Html.get.datalink <data_svar>

A datalink provides a method for sending a message from an HTML program to the BASIC! programmer. There are two parts to a datalink in an HTML file:

1. The JavaScript that defines the datalink function
2. The HTML code that calls the datalink function.

The BASIC! Program requires a mechanism for communicating with a website's HTML code.

Html.get.datalink gets the next datalink string from the datalink buffer. If there is no data available then the returned data will be an empty string (""). You should program a loop waiting for data:

```
Do
 Html.get.datalink data$
Until data$ <> ""
```

The returned data string will always start with a specific set of four characters—three alphabetic characters followed by a colon (":"). These four characters identify the return datalink data type. Most of the type codes are followed by some sort of data. The codes are:

BAK: The user has tapped the BACK key. The data is either "1" or "0".

If the data is "0" then the user tapped BACK in the start screen. Going back is not possible therefore HTML has been closed.

If the data is "1" then going back is possible. The BASIC! programmer should issue the command **Html.go.back** if going back is desired.

LNK: The user has tapped a hyperlink. The linked-to URL is returned. The transfer to the new URL has not been done. The BASIC! programmer must execute an **Html.load.url** with the returned URL (or some other URL) for a transfer to occur.

ERR: Some sort of fatal error has occurred. The error condition will be returned. This error code always closes the HTML engine. The BASIC! output console will be displayed.

FOR: The user has tapped the Submit button on a form with action='FORM' The form name/value pairs are returned.

DNL: The user has clicked a link that requires a download. The download URL is supplied. It is up to the BASIC! programmer to do the download.

DAT: The user has performed some action that has caused some JavaScript code to send data to BASIC! by means of the datalink. The JavaScript function for sending the data is:

```
<script type="text/javascript">
  function doDataLink(data) {
    Android.dataLink(data);
  }
</script>
```

STT: Calls the speech recognition. Note, for detection the first four characters are used.

EJS: Returns the result of **Html.evaluate.js** if any.

CME: Returns a Console Message as String of type

```
CME:<lineNumber>,<sourceId>,<messageLevel>,<message>
```

unless turned off by an **Html.open** command option. Some devices run into trouble in conjunction with GW-Lib and CME.

30.6 Html.go.back

Syntax: Html.go.back

Go back one HTML screen, if possible.

30.7 Html.go.forward

Syntax: Html.go.forward

Go forward one HTML screen, if possible.

30.8 Html.load.string

Syntax: Html.load.string <html_sexp>

Loads and displays the HTML contained in the string expression. The base page for this HTML will be:

```
<pref base drive>/rfo-basic/data/
```

30.9 Html.load.url

Syntax: Html.load.url <file_sexp>

Loads and displays the file specified in the string <file_sexp>. The file may reside on the Internet or on your Android device. In either case, the entire URL must be specified.

The command:

```
Html.load.url "http://laughton.com/basic/"
```

will load and display the BASIC! home page.

The command:

```
Html.load.url "htmlDemo1.html"
```

will load and display the HTML file "htmlDemo1.html" residing in BASIC!'s default "data" directory, as set by your "Base Drive" preference. You may also use a fully-qualified pathname. With the default "Base Drive" setting, this command loads the same file:

```
Html.load.url "file:///sdcard/rfo-basic/data/htmlDemo1.html"
```

When you tap the BACK key on the originally-loaded page, the HTML viewer will be closed and the BASIC! output console will be displayed. If the page that was originally loaded links to another page and then the BACK key is tapped, it will be up to the BASIC! programmer to decide what to do.

This command can also be used to execute JavaScripts that were included in the HTML file that is currently displayed. For example, the command:

```
Html.load.url "javascript:myFunction();"
```

will call the JavaScript function myFunction(). The function must already exist in the HTML that was previously loaded via Html.load.string or Html.load.url. For example:

```
<html>
<head>
 <script>
  function myFunction() {
   ...
  }
 </script>
</head>
...
</html>
```

If you need to pass parameters to the function, they must be included as strings:

BASIC!:

```
Html.load.url "javascript:myFunction('123');"
```

HTML:

```
<html>
<head>
 <script>
  function myFunction(Parm) {
   ...
  }
 </script>
</head>
...
</html>
```

Note that JavaScript will accept single-quotes (') as well as double-quotes (").

If you need to pass parameters that are already in BASIC! variables, you have to include them as part of the string:

BASIC!:

```
Alfa = 123
Beta$ = "abc"

Html.load.url "javascript:myFunction('" + str$(Alfa) + ~
              "', '" + Beta$ + "');"
```

HTML:

```
<html>
<head>
 <script>
  function myFunction(Parm1, Parm2) {
   ...
  }
 </script>
</head>
...
</html>
```

Note that numeric variables must be converted to strings to be passed as part of one big string. In the above example, the string will become:

```
"javascript:myFunction('123', 'abc');"
```

30.10 Html.onHtmlReturn.resume

Syntax: Html.onHtmlReturn.resume

This statement should be placed at the end of the interrupt handler at **OnHtmlReturn:.**

30.11 Html.open

Syntax: Html.open {{{{<Title_nexp>}, <Orientation_nexp>}, <securityLevel_nexp>}, <CME_nexp>}

This command must be executed before using the HTML interface.

The Status Bar will be shown on the Web Screen if <Title_nexp> is 1. If the <Title_nexp> is not present, the Status Bar will not be shown. If <Title_nexp> is negative, **Html.open** expects a layout bundle defined by items in the following table.

Upon opening the HTML screen, the orientation will be determined by the <Orientation_nexp> value. <Orientation_nexp> values are the same as values for the **Html.orientation** command. If the <Orientation_nexp> is not present, the default orientation is determined by the orientation of the device.

Both <Title_nexp> and <Orientation_nexp> are optional; however, <Title_nexp> must be present in order to specify <Orientation_nexp>.

Executing a second **Html.open** before executing **Html.close** will generate a run-time error.

A security level can be set by <securityLevel_nexp> as follows:

>= 4 Javascript disabled
< 4 Javascript enabled
< 3 JavaScript can open windows automatically
< 2 Allow JavaScript file access from file URLs
< 1 Allow JavaScript to access the contents of file URLs from any origin.

The default value is 3.

To suppress returning a Console Message, set <CME_nexp> to 0. This can be useful, especially if you are using GW-Lib. The default value is 1.

The options bundle at -<Title_nexp> controls the layouts of the Action and Navigation bars as follows:

Table of Layout Control Options		
Key	**Value**	**Description**
_ShowActionbar	0 or 1 (numeric)	If 1 Show the Action bar if it is not currently showing. It is needed to show titles and to change the background color of the Statusbar. If 0 (default) Hide the Actionbar if it is currently activated.
_Title	String by default	Set the action bar's title.
_Subtitle	String by default	Set the action bar's subtitle.
_TitleShow	0 or 1 (numeric)	If 1 (default) Show the Action bar if it is not currently showing. It will resize application content to fit the new space available. If 0 Hide the Actionbar if it is currently showing. It will resize application content to fit the new space available.

Table of Layout Control Options		
Key	**Value**	**Description**
_TitleIcon	Icon file path	Add a large icon to the notification content view. http://romannurik.github.io/AndroidAssetStudio/index.html
_TitleHomeEnabled	0 or 1 (numeric)	Set whether to include the application home accordance in the action bar. Home is presented as an activity icon. Have to be 1 if you want to show the icon. Have to be 0 if you want to hide the icon. The default setting is API dependent.
_TitleBackground	Background file path	
_TitleHtml	0 or 1 (numeric)	Returns displayable styled text from the provided HTML string. But not all tags are supported. Uses parts of TagSoup library to handle real HTML, including all of the brokenness found in the wild. <b> <big> <font size="..." color="..." face="..."> <h1>, <h2>, <h3>, <h4>, <h5>, <h6> <i> <small> <strike>? < A.7 <sub> <sup> <tt>? <u> Replace Space with , & with &amp, < with &lt, > with &gt, " with &quot if necessary. Usable for Title and Subtitle. Keep in mind that the Actionbar height will not be expanded.
_ShowStatusbar	0, 1 or 2 (numeric)	If 1 (default) The Status bar will be displayed. If 2 The Status bar will be transparent displayed. Min. Lollipop 5.0 (API 21) If 0 The Status bar will be hidden to the background. Min. Nougat 7.0 (API 24) Will be switched to option 2 or 1 if the current API level is lower.
_StatusbarColor	{Alpha,}Red,Green,Blue (comma delimited string) or _{Alpha,}ColorName	Min. Lollipop 5.0 (API 21) Note, the Actionbar has to be activated by

Table of Layout Control Options		
Key	**Value**	**Description**
	({comma delim.} string) or #{hn}hnhnhn (hex. string)	**_ShowActionbar.**
_StatusbarLight	0 or 1 (numeric)	If 0 (default) The Status bar background is dark. In this case the **bar content** will be **light**. If 1 The Status bar background is light. In this case the **bar content** will be **dark**. Min. Lollipop 5.0 (API 21)
_ShowNavigationbar	0, 1 or 2 (numeric)	If 1 (default) The Navigation bar will be displayed. If 2 The Navigation bar will be transparent displayed. Min. Lollipop 5.0 (API 21) If 0 The Navigation bar will be hidden to the background. Min. Nougat 7.0 (API 24) Will be switched to option 2 or 1 if the current API level is lower.
_NavigationbarColor	{Alpha,}Red,Green,Blue (comma delimited string) or _{Alpha,}ColorName ({comma delim.} string) or #{hn}hnhnhn (hex. string)	Min. Lollipop 5.0 (API 21)
_NavigationbarLight	0 or 1 (numeric)	If 0 (default) The Navigation bar background is dark. In this case the **bar content** will be **light**. If 1 The Navigation bar background is light. In this case the **bar content** will be **dark**. Min. Lollipop 5.0 (API 21)
_Menu	Menu Bundle Pointer	Creates menu entries. A successful selection will be returned as a human readable JSON string. See the example at Console.title for more details.

30.12 Html.orientation

Syntax: Html.orientation <nexp>

The value of the <nexp> sets the orientation of screen as follows:

-1 = Orientation depends upon the sensors.
0 = Orientation is forced to Landscape.
1 = Orientation is forced to Portrait.
2 = Orientation is forced to Reverse Landscape.
3 = Orientation is forced to Reverse Portrait.

30.13 Html.paperformats

Syntax: Html.paperformats <bundlePointer_nexp>

Returns possible paper formats as a bundle. The bundle key is also the paper format key name. The bundle entry returns more information about the specified format as a delimited string.

30.14 Html.post

Syntax: Html.post <url_sexp>, <list_nexp>

Execute a Post command to an Internet location.

<url_sexp> is a string expression giving the url that will accept the Post.

<list_nexp> is a pointer to a string list which contains the Name/Value pairs needed for the Post.

30.15 Html.screenshot

Syntax: Html.screenshot <filename_sexp>

Saves the current screen to a file. The default path is "<pref base drive>/rfo-basic/data/".

The file will be saved as a JPEG file if the filename ends in ".jpg".

The file will be saved as a PNG file if the filename ends in anything else (including ".png").

30.16 Html.to.pdf

Syntax: Html.to.pdf <filename_sexp> {{{{, <paperformat_sexp>}, <orientation_sexp>}, <resolution_nexp>}, <color_sexp>}

Saves the current HTML content to a file. The default path is "<pref base drive>/rfo-basic/data/".

The chosen paper format will be set by <paperformat_sexp>. Default is "ISO_A4".

The optional <orientation_sexp> defines the sheet orientation. "L" is landscape and "P" the default portrait. The resolution can be set by <resolution_nexp> in dots per inch. The default setting is 600 dpi. By default <color_sexp> is set to "C" for color output. "M" saves the content monochrome.

Note that to overwrite, you have to delete the old PDF file first.

Example:

```
Html.open 0, -1
!Html.load.url "htmlDemo1.html"
mM$ = "Hello<br>World"
Html.load.string mM$
Html.paperFormats pfb
Bundle.keys pfb, pfl
Dialog.select sel, pfl, "Choose a paper format"
List.get pfl, sel, pf$
fileName$ = "html.pdf"
File.exists fOk, fileName$
If fOk Then File.delete dOk, fileName$
Html.to.pdf fileName$, pf$ , "P"
Do % Needed, because asynchronous background task.
 Pause 100
 File.exists fOk, fileName$
Until fOk
Html.close
```

30.17 Is_Html

Syntax: Is_Html()

Is_Html() is a function which returns the HTML mode status. If 1, the HTML mode is open. If 0, the HTML mode is not open.

30.18 OnHtmlReturn:

Syntax: OnHtmlReturn:

Label for an interrupt handler which traps arrival of HTML data. When done, execute the **Html.onHtmlReturn.resume** statement to resume the interrupted program.

31 Infrared Port Commands

If your device supports sending infrared codes, you can test for success by using a smartphone camera.

For more information search Youtube for:

#171 Arduino Guide to Infrared (IR) Communication also for ESP32 and ESP8266

https://github.com/SensorsIot/Arduino-IRremote

31.1 IrPort

Syntax: IrPort <success_svar>, <frequency_nexp>, <pattern_array[]>

The attribute <frequency_nexp> specifies the carrier frequency. 38 kHz is a common one. The array <pattern_array[]> contains the pattern pairs. The first element of the pair sets the IR LED for maybe 150 µs to on and the second element switches the LED for maybe 40 µs to off. The whole duration of the pattern array has not be longer than 2,000,000 µs (2 sec).

If the sending is successfully <success_svar> returns an empty string.

Otherwise following messages will be returned:

- The whole duration of the pattern array has not be longer than 2,000,000 µs (2 sec)
- No Ir-Emitter found
- The array of pattern must have even number of elements

32 Internet Access Commands

32.1 Http.post

Syntax: Http.post <url_sexp>, <list_nexp>, <result_svar> {{{{{, <ok_svar>}, <use_caches_lexp>}, <charset_sexp>}, <connect_timeout_nexp>}, <read_timeout_nexp>}

Execute a Post command to an Internet location.

<url_sexp> contains the url ("http://....") that will accept the Post.

<list_nexp> is a pointer to a string list which contains the Name/Value pairs needed for the Post.

<result_svar> is where the Post response will be placed.

<url_sexp> contains the url ("http(s)://....") that will accept the Post.

If the server returns an error <ok_svar> will not contain "ok". An error handler has to be placed behind this command, if needed.

If you want a cache <use_caches_lexp> has to be > 0.

To choose a character set use <charset_sexp>. Default is "_UTF-8" for text, else use "_ISO-8859-1" for binary data.

If you want to specify a connect timeout, use <connect_timeout_nexp> with a value > 0.

If you want to specify a read timeout, use <read_timeout_nexp> with a value > 0.

32.2 Http.request

Syntax: Http.request <request_type_sexp>, <url_sexp>, <list_nexp>, <result_svar>{{{{{, <ok_svar>}, <use_caches_lexp>}, <charset_sexp>}, <connect_timeout_nexp>}, <read_timeout_nexp>}

Execute a request command to an Internet location.

Available request types are "_DELETE", "_GET", "_PATCH", "_POST", "_PUT" with write and read access and "_HEAD", "_OPTIONS", "_TRACE" with read-only access.

<url_sexp> contains the url ("http(s)://....") that will accept the request.

<list_nexp> is a pointer to a string list which contains the Name/Value pairs needed for the request. If you have nothing to write, create an empty list. <result_svar> is where the request response will be placed.

If the server returns an error <ok_svar> will not contain "ok". An error handler has to be placed behind this command, if needed.

If you want a cache, <use_caches_lexp> has to be > 0.

To choose a character set use <charset_sexp>. Default is "_UTF-8" for text, else use "_ISO-8859-1" for binary data.

If you want to specify a connect timeout, use <connect_timeout_nexp> with a value > 0.

If you want to specify a read timeout use, <read_timeout_nexp> with a value > 0.

33 Interrupts, Event Handlers and Errors

You can perform physical actions that tell your program to do something. When you touch the screen or press a key you cause an *event*. These events are *asynchronous*, that is, they happen at times your program cannot predict. BASIC! detects some events so your program can respond to them.

BASIC! handles events as *interrupts*. Each event that BASIC! recognizes has a unique *Interrupt Label*. When an event occurs, BASIC! looks for the Interrupt Label that matches the event.

- If you have not written that Interrupt Label into your program, the event is ignored and your program goes on running as if nothing happened.
- If you have included the right Interrupt Label for the event, BASIC! jumps to that label and continues execution at the line after the label. This is called trapping the event.

BASIC! does not necessarily respond to the event as soon as it occurs. The statement that is executing when the event occurs is allowed to complete, then BASIC! jumps to the Interrupt Label.

When you use an Interrupt Label to trap an event, BASIC! executes instructions until it finds a *Resume* statement that matches the Interrupt Label. During that time, it records other events but it does not respond to them. The block of code between the Interrupt Label and the matching Resume statement may be called an *Interrupt Service Routine (ISR)* or, if you prefer, an *Event Handler*.

When BASIC! executes the event's Resume statement, it resumes normal execution.

- BASIC! jumps back to where it was running when the interrupt occurred.
- BASIC! again responds to other events, including any that occurred while it was handling an event. In other words events are be put in a waiting queue. Thus many events can be triggered and will be handled on the first-in-first-out principle, but **OnError:** is always serviced first.

An Interrupt Label looks and behaves just like any other label in BASIC!. However, you must not execute any of the Resume statements except to finish an event's handler.

Variables in functions are by default local. So if an interrupt occurs while your code is executing a function, you might get into trouble. This can be solved by the use of **Globals.all** at the start of an ISR (right after the OnXxx: statement) and **Globals.none** just before the **Resume** statement.

For interrupt handling to be as fast as possible:

- Use a timer interrupt instead of **Pause** in your main loop. This solves a lot of interrupt issues, because **Pause** also delays the interrupts.
- You can also use the **Sched.set** command as second timer.

33.1 Interrupt Labels

The following events can be trapped:

Interrupt Label	Resume Statement
OnBackground:	Background.resume
OnBackKey:	Back.resume
OnBroadcast:	Error: Reference source not found
OnBtReadReady:	Bt.onReadReady.resume
OnBtStatus:	Bt.onStatus.resume
OnConsoleTouch:	ConsoleTouch.resume
OnError:	None

Interrupt Label	Resume Statement
OnGrScreen:	GR.onGrScreen.resume
OnGrTouch:	Gr.onGrTouch.resume
OnGrTouchMove:	Gr.onGrTouchMove.resume
OnGrTouchUp:	Gr.onGrTouchUp.resume
OnHtmlReturn:	Html.onHtmlReturn.resume
OnKbChange:	Kb.resume
OnKeyDown:	KeyDown.resume
OnKeyPress:	Key.resume
OnLowMemory:	LowMemory.resume
OnMenuItem:	MenuItem.resume
OnMenuKey:	MenuKey.resume
OnTimer:	Timer.resume
OnUsbReadReady:	Usb.onReadReady.resume
OnUsbStatus:	Usb.onStatus.resume

Most of the above interrupt labels and their Resume statements are described in various sections of this manual. Some do not have appropriate sections, and are described here.

33.2 Back.resume

Syntax: Back.resume

This statement should be placed at the end of the interrupt handler at **OnBackKey:**.

33.3 GetError$

Syntax: <svar> = GetError$()

Function that returns information about a possible error condition.

An error that stops your program writes an error message to the Console. If you trap the error with the **OnError:** interrupt label, your program does not stop and the error is not printed. You can use **GetError$()** to retrieve the error message.

Certain commands can report errors without stopping your program. These commands include **App.broadcast**, **App.start**, **Audio.load**, **Byte.open**, **Text.open**, **Zip.Open**, **Encrypt**, **Decrypt**, **Font.load**, **GPS.open**, **GrabFile**, **GrabURL**, **Gr.get.type**, the **Encode$()** and **Decode$()** functions, and any command that can create a bitmap.

When you run one of these commands, you can call **GetError$()** to retrieve error information. For example, if **Text.open** cannot open a file, it sets the file pointer to -1, and writes a **GetError$()** message such as "<filename> not found". If no error occurred, **GetError$()** returns "No error".

Because there are commands that clear the error message, you should not expect **GetError$()** to retain its message. Capture the message in a variable as soon as possible, and do not call **GetError$()** again for the same error.

Example:

```
% Function Exeption Handler
Fn.def myFunc()
 func$ = "myFunc"
 p = p + 4 : z = 7 : k = 5
```

```
 p$ = p % Line with Error code.
 labelGoOn_myFunc:
 Print p, z, k
Fn.end

p = 23
z = 44
k = 99
myFunc()
End

OnError:
If func$ = "myFunc" Then
 Print GetError$()
 ! Globals.fnimp z % Globals.fnimp, Globals.all and Globals.none
 ! together are not possible in the same function
 Print p     % returns 4, because still in the function myFunc
 Print z     % returns 44, because imported in the area of the function
 Print k     % returns 0, because not imported
 Globals.ALL % Usefull for exeption handling,
             % the value of the variable is not imported!
 Print k     % returns 99, because access to the global symbol table
 Globals.none
 GoTo labelGoOn_myFunc
EndIf
```

33.4 LowMemory.resume

Syntax: LowMemory.resume

This statement should be placed at the end of the interrupt handler at **OnLowMemory:**.

33.5 MenuItem.resume

Syntax: MenuItem.resume

This statement should be placed at the end of the interrupt handler at **OnMenuItem:**.

33.6 MenuKey.resume

Syntax: MenuKey.resume

This statement should be placed at the end of the interrupt handler at **OnMenuKey:**.

33.7 OnBackKey:

Syntax: OnBackKey:

Label for an interrupt handler which traps the BACK key.

If a program does not have an **OnBackKey:** label, the BACK key will normally halt program execution.

If you trap the BACK key with **OnBackKey:**, the BACK key does not stop your program. You should either terminate the run in the **OnBackKey:** code or provide another way for the user to tell your program to stop, especially if the program is in Graphics mode where there is no menu. If you do not, then there will be no stopping the program (other than using Android Settings or a task killer application).

33.8 OnError:

Syntax: OnError:

Label for a special interrupt handler which traps a run-time error as if it were an event, except that:

- **OnError:** has no matching Resume command. You can use **GoTo** to jump anywhere in your program.
- **OnError:** is not locked out by other interrupts.
- **OnError:** does not lock out other interrupts.

If a program does not have an **OnError:** label and an error occurs while the program is running, an error message is printed to the Output Console and the program stops running.

If the program does have an **OnError:** label, it will not stop on an error. Instead, it jumps to the **OnError:** label just like a **GoTo** statement would. The error message is not printed, but it can be retrieved by the **GetError$()** function. If the error occurs inside a user defined function, all of the function's variables are local to the function and not available to the ISR.

Be careful. An infinite loop will occur if a run-time error occurs within the **OnError:** code. You should not place an **OnError:** label into your program until the program is fully debugged. Premature use of **OnError:** will make the program difficult to debug.

33.9 OnLowMemory:

Label for an interrupt handler which traps the Android "low memory" warning.

If Android is running out of memory, it may kill applications running in the background, but first it will broadcast a "low memory" warning to all applications. If you do not have an **OnLowMemory:** label in your program, you will see "Warning: Low Memory" printed on the Console.

33.10 OnMenuItem:

Label for an interrupt handler which traps a menu item being selected.

33.11 OnMenuKey:

Syntax: OnMenuKey:

Label for an interrupt handler which traps the MENU key.

Note: This interrupt does not work unless your device has a MENU key. Android devices since Honeycomb typically do not have a MENU key.

34 JSON Commands

JSON ("**J**ava**S**cript **O**bject **N**otation") defines a lightweight data format in which information such as objects, arrays, and other variables can be stored in readable form. JSON strings can be transformed to bundles with corresponding key strings, or converted to XML strings.

JSON is often used to easily exchange information between client and server and is a handy alternative to XML.

See the documentation at https://www.w3schools.com/js/js_json_intro.asp.

A JSON object begins with a "{" and ends with a "}".

Workaround for a known bundle put issue:

```
GrabFile j$, "project.json"

% Fill empty Arrays with a "" member.
j$ = Replace$(j$, "[]", "[" + "\"" + "\"" +"]")

! Now fill empty Strings with "n.a.m." = "not a member".
j$ = Replace$(j$, "\"" + "\"", "\"" + "n.a.m."+ "\"")
```

34.1 Is_Json

Syntax: Is_Json(<json_sexp>)

This function checks if <json_sexp> can be parsed as a JSON string.

Returns 1 if true, 0 if false.

See also: **JsonToXml$()**, **Bundle.GJ**

34.2 XmlToJson$

Syntax: XmlToJson$ (<xml_sexp>{, <spaces_nexp>})

Retruns a JSON string converted from an XML string.

Needs a parseable JSON string.

If the optional **<spaces_nexp>** returns a number > -1, a structural printout is delivered.

The number of spaces defines the tabulator distance.

If **<spaces_nexp>** is -1 or not given, a compact printout is returned.

There may be some post-processing needed, especially if JSON objects beginning with a "{" and ending with a "}" are involved.

If an error occurred, an empty string will be returned.

See also: **JsonToXml$()**, **Bundle.PJ**

HTML Example:

```
<!DOCTYPE html>
<html>
<body>

<h2>Store and retrieve data from local storage.</h2>
```

```
<p id="JsonOut"></p>
<p id="lastNameOut"></p>
<p id="postalCodeOut"></p>

<script>
var myObj, myJSON, text, obj, obj2;
//Storing data:
myObj = {
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
};
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);
document.getElementById("JsonOut").innerHTML = myJSON;
// Retrieving data:
text = localStorage.getItem("testJSON");
// First level
obj = JSON.parse(text);
document.getElementById("lastNameOut").innerHTML = obj.lastName;
// Second level
obj2 = obj.address;
document.getElementById("postalCodeOut").innerHTML = obj2.postalCode;
</script>

</body>
</html>
```

OliBasic Example:

```
! Stringified JSON object
myJSON$ = "{\"firstName\": \"John\",\"lastName\": \"Smith\",\"age\": 25," +~
"\"address\": {\"streetAddress\": \"21 2nd Street\",\"city\": \"New York\"," +~
"\"state\": \"NY\",\"postalCode\": 10021},\"phoneNumbers\": " +~
"[{\"type\": \"home\",\"number\": \"212 555-1234\"}," +~
"{\"type\": \"fax\",\"number\": \"646 555-4567\" }] }"
Print "JsonOut:", myJSON$

Print "Is_json", IS_JSON( myJSON$)

! First level
! Workaround for a known bundle put issue:
! Fill empty Arrays with a "" member.
myJSON$ = Replace$(myJSON$, "[]", "[" + "\"" + "\"" +"]")
! Now fill empty Strings with  n.a.m. = "not a member".
myJSON$ = Replace$(myJSON$, "\"" + "\"", "\"" + "n.a.m."+ "\"")
Bundle.PJ testJSON, myJSON$
Bundle.get testJSON, "lastName", lastName$
Print "LastNameOut:", lastName$
Bundle.get testJSON, "address", obj_address$
```

```
! Second level
Print "Is_json", IS_JSON( obj_address$)
Bundle.PJ obj_address, obj_address$
Bundle.get obj_address, "postalCode", postalCode$
Print "PostalCodeOut:", postalCode$

! Now all backwards
postalCode$ = "10028"
Print "The changed PostalCode:", postalCode$
Bundle.put obj_address, "postalCode", postalCode$
Bundle.GJ obj_address, obj_address$

! Back to the first level
Bundle.put testJSON, "address", obj_address$
Bundle.put testJSON, "lastName", lastName$
Bundle.GJ testJSON, myJSON$
Print "JsonIn:", myJSON$

! You see we get all and the changed postal code back.
! Because we do not change the Bundle pointer (IDs).
! The order within the JSON text may have changed.
! A good JSON interpreter should solve this.
```

35 List Commands

A List is similar to a single-dimension array. The difference is in the way a List is built and used. An array must be dimensioned before being used. The number of elements to be placed in the array must be predetermined. A List starts out empty and grows as needed. Elements can be removed, replaced and inserted anywhere within the list.

There is no fixed limit on the size or number of lists. You are limited only by the memory of your device.

Another important difference is that a List is not a variable type. A numeric pointer is returned when a list is created. All further access to the List is by means of that numeric pointer. One implication of this is that it is easy to make a List of Lists. A List of Lists is nothing more than a numeric list containing numeric pointers to other lists.

Lists may be copied into new Arrays. Arrays may be added to Lists.

All of the List commands are demonstrated in the Sample Program file, **f27_list.bas**.

35.1 List.add

Syntax: List.add <pointer_nexp>{, <exp>}...

Adds the values of the expressions <exp>... to specified list. The expressions must all be the same type (numeric or string) as the list.

The list of <exp>s may be continued onto the next line by ending the line with the "~" character. The "~" character may be used between <exp> parameters, where a comma would normally appear. The "~" itself separates the parameters; the comma is optional.

The "~" character may not be used to split a parameter across multiple lines.

Examples:

```
List.add Nlist, 2, 4, 8 , n^2, 32

List.add Hours, 3, 4,7,0, 99, 3, 66~  % comma not required before ~
                37, 66, 43, 83,~      % comma is allowed before ~
                83, n*5, q/2 +j

List.add Name~
"Bill", "Jones"~
"James", "Barnes"~
"Jill", "Hanson"
```

35.2 List.add.list

Syntax: List.add.list <destination_list_pointer_nexp>, <source_list_pointer_nexp>{{, <sub_list_start_nexp>}, <sub_list_end_nexp>}

Appends the elements in the source list to the end of the destination list. The two lists must be of the same type (string or numeric).

Optionally, a sub list can be added by specifying the start and end indices in <sub_list_start_nexp> and <sub_list_end_nexp>. If only <sub_list_start_nexp> is given, only the value at this index will be added.

35.3 List.add.array

Syntax: List.add.array <list_pointer_nexp>, Array[{<start>,<length>}]

Appends the elements of the specified array (Array[]) or array segment (Array[start,length]) to the end of the specified list.

The Array type must be the same as the list type (string or numeric).

35.4 List.binary.search

Syntax: List.binary.search <pointer_nexp>, search_nexp|search_sexp, <result_nvar>

Searches the specified list for the specified string or numeric value. The position of the first (left-most) occurrence is returned in the numeric variable <result_nvar>. If the value is not found in the list, then the result is zero.

This command uses the very fast binary-search-method. **Keep in mind, that you have to sort the list before to prevent unexpected results.**

Example:

```
List.sort lPtr, 0, "fr_FR"
List.binary.search lPtr, "Paris", resultIndex
```

35.5 List.bounds.2d

Syntax: List.bounds.2d <pointer_nexp>, <xMin_nvar>, <yMin_nvar>, <xMax_nvar>, <yMax_nvar>

Gets the bounding rectangle of an xy list that can be used in conjunction with gr.polygon.

For a complete 2D operation, you need x and y values, which is an even number of values.

If the number of list items is not even, the last value will be ignored.

Example:

```
List.create n, ln1
List.add ln1, 100, 100, 200, 100
List.add ln1, 200, 200, 100, 200

mvX = 0: mvY = 0 % Move x,y for testing
LIST.create n, ln2
List.add ln2, mvX + 60, mvY + 150, mvX + 150, mvY + 60
List.add ln2, mvX + 240, mvY + 150, mvX + 150, mvY + 240

List.bounds.2D ln1, left, top, right, bottom
Print "ln1:", left, top, right, bottom
Array.load r1[], left, top, right, bottom % Result 1

List.bounds.2D ln2, left, top, right, bottom
Print "ln2:", left, top, right, bottom
Array.load r2[], left, top, right, bottom % Result 2

left = 1: top = 2: right = 3: bottom = 4
collision = 1
If ((r1[bottom] < r2[top]) | (r2[bottom] < r1[top]) | ~
 (r1[right] < r2[left]) | (r2[right] < r1[left])) ~
 Then collision = 0 % Test for collision
Print "collision: "; collision

Pause 3000

Gr.open "_White", 1 , 1
Gr.color "_Red"
Gr.poly obj1, ln1, 0, 0
```

```
Gr.color "_Green", 0
Gr.poly obj2, ln2, 0, 0
Gr.render
```

35.6 List.bounds.3d

Syntax: List.bounds.3d <pointer_nexp>, <xMin_nvar>, <yMin_nvar>, <zMin_nvar>, <xMax_nvar>, <yMax_nvar>, <zMax_nvar>

Gets the bounding box of an xyz list.

For a complete 3D operation, you need x, y and z values. When the number of values is divided by 3, the result must be an integer. If a 3D vector is not complete, it will be ignored.

35.7 List.clear

Syntax: List.clear <pointer_nexp>{, <pointer_nvar>}...

Clears the list pointed to by the list pointer and sets the list's size to zero. Optionally, several lists can be cleared.

35.8 List.create

Syntax: List.create N|S, <pointer_nvar>{, <pointer_nvar>}...

Creates a new, empty list of the type specified by the N or S parameter. A list of strings will be created if the parameter is **S**. A list of numbers will be created if the parameter is **N**. Do not put quotation marks around the N or S.

The pointer to the new list will be returned in the <pointer_nvar> variable.

The newly created list is empty. The size returned for a newly created list is zero.

If a <pointer_nvar> specifies an existing list, it will be cleared and overwritten without warning!

Optionally, this command accepts more than one list of the same type.

35.9 List.dimsort.by

Syntax: List.dimsort.by <sorted_nexp>, <toSort_nexp>, <dim_s_nexp>, <by_nexp>, <dim_b_nexp>, <which_nexp>{{, <sort_mode_nexp>}, <exclude_sexp>, <exValue_sexp>}

The returned numeric list <sorted_nexp> is a copy of the numeric list <toSort_nexp>, but it is ordered by the internally sorting index. Its <dim_s_nexp> specifies the length of the entry sets, which are sorted. The internal sorting index is created by the numeric list <by_nexp>. Its <dim_b_nexp> specifies the length of the entry sets that contain the entry that forms the basis of the sort. The argument <which_nexp> specifies the member of each set which is the basis.

Options of <sort_mode_nexp>:

- -1 Nothing is sorted, but a copy under conditions is optional possible.
- 0 Sorted in ascending order (default)
- 1 Sorted in descending order

If the optional <exclude_sexp> specifies a condition, under which it is possible to exclude an entry set. Conditions are _=, _<>, _<, _>, _<=, _>= as Strings. The value to be compared to is given by <exValue_sexp>.

Example:

```
List.create n, xy, theResult
List.add xy, 15, 5, 44, 4, 23, 3, 12, 2, 14, 1 % Two dimensional xy List
List.dimsort.by theResult, xy, 2, xy, 2, 2, 0, "_<", 5
! The List theResult returns 14, 1,   12, 2,   23, 3,   44, 4 sorted by the y
entries.
List.row.print theResult, 2
```

35.10 List.get

Syntax: List.get <pointer_nexp>, <index_nexp>, <var>{, <index_nexp>, <var>}...

The list element specified by <index_nexp> in the list pointed to by <pointer_nexp> is returned in the specified string or numeric variable <var>. The index is one-based. The first element of the list is 1.

The return element variable type must match the list type (string or numeric). Optionally, several elements can be retrieved.

Example:

```
List.size xyzPoints, ls
For u = 1 To ls Step 3
 List.get xyzPoints, u, x, ++u, y, ++u, z % ++u equals u = u + 1
 Print x, y, z
Next
```

35.11 List.insert

Syntax: List.insert <pointer_nexp>, <index_nexp>, <sexp>|<nexp>

Inserts the <sexp> or <nexp> value into the list pointed to by <pointer_nexp> at the index point <index_nexp>. If the index point is one more than the current size of the list, the new item is added at the end of the list.

The index is ones based. The first element of the list is 1.

The inserted element type must match the list type (string or numeric).

35.12 List.join

Syntax: List.join <result_nexp>, {<scr_left_nexp>|<scr_left_sexp>}, {<scr_right_nexp>|<scr_right_sexp>}, <delim_sexp> {{{{, <_oper_arg_sexp>}, <start_nexp>}, <end_nexp>}, <add_nexp>}

Joins the optional source lists <scr_left_nexp> and <scr_right_nexp> into the list <result_nexp> item by item. The list <scr_right_nexp> is optional.

With your chosen list type (S or N) you control the type of your output <result_nexp> automatically. The type of source lists is recognized and converted internally and automatically into strings.

The optional <scr_left_sexp> and <scr_right_sexp> represent a list with limited size. The size limits depend on the size of the other list or the arguments <end_nexp> and <add_nexp>. Numeric values have to be converted to strings, maybe with **STR$**(<nexp>).

If the item count of the lists <scr_left_nexp> and <scr_right_nexp> is different, the item count of the list with the most items or the setting is used. In this case the empty item returns "" or 0.0 (_+(.), _-(.)) or 1.0 (_*(.), _/(.)([0-9])). T**ry two lists with different lengths to see the results!** Compare with the example line which ends with **!*/!.**

The delimiter <delim_sexp> is added after the <scr_left_nexp> item.

After that, the optional <_oper_arg_sexp> is executed before adding into the list <result_nexp>.

The parameters <start_nexp> and <end_nexp> set the range of the source list to work with. If no values are given, the entire list is used.

Remember that <end_nexp> is **not limited!** See the example line which ends with "An index list function".

The <add_nexp> argument lets you append (<add_nexp> = 1) the results to the output lists. If this argument is 0 (default) the output lists are cleared before execution.

Operators

For String expressions valid operators are: _+$

For Numeric expressions valid operators are: _+, _-, _*, _/{scale_nexp},

_*sin, _*cos, _*tan, _*asin, _*acos, _*atan, _*sqr

You can also put a point (.) into the operator to account for zeros (0) before missed decimal points like , _+., _-., _*., _/.{scale_nexp}, _*.sin …. .

The optional {scale_nexp} is set per default to 16.

_+i equals the list index and _-i equals the list index as a negative numper

_atan4 is special, because it returns the angle for the term y / x in the range [0° … 360°]

counterclockwise. <scr_**left**_nexp> is **y!!!,** <scr_**right**_nexp> is **x.**

Note, if y = 0 and x = 0, the result is 0.

_min, _max are special, because it returns the minimum or maximum of <scr_**left**_nexp> and <scr_**right**_nexp> is **x.**

The four basic arithmetic operations are calculated as BigDecimal, after the entire input has been converted to strings before. Other operations will be computed as Double.

Keep in mind, that values from type Double contain only maximal 15 **correct** digits.

Trigonometric functions use **degrees** as in- and output.

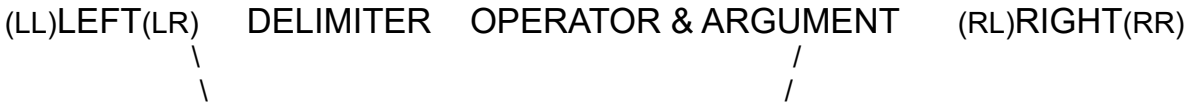
If the <_oper_arg_sexp> starts additionally with **_D** all arithmetic operations are calculated as Double, that increases the speed round about 130% with lost of accuracy. In this case the scale argument has no effect.

Arguments

Arguments as string expressions have to be enclosed in quotation marks: "text"

For example:

```
\"text\"  or  CHR$(34) + "text" + CHR$(34)
```



\ /

(OL)OUTPUT RESULT(OR)

Examples:

```
a <scr_left_nexp> item = 20
a <scr_rigth_nexp> item = 010
<delim_sexp> = "." and  <_oper_arg_sexp> = ""           returns 20.010
<delim_sexp> = ""  and  <_oper_arg_sexp> = "_+"         returns 30
<delim_sexp> = "." and  <_oper_arg_sexp> = "_+"         returns 20.1
<delim_sexp> = ""  and  <_oper_arg_sexp> = "_+."        returns 20.01
<delim_sexp> = ""  and  <_oper_arg_sexp> = "_/.3"       returns 2000.000
<delim_sexp> = ""  and  <_oper_arg_sexp> = "_min."      returns 0.01
<delim_sexp> = "§" and  <_oper_arg_sexp> = "_+$_" + "#" returns #20§010#

only <scr_left_nexp> item = 20
<delim_sexp> = ""  and  <_oper_arg_sexp> = "_-._04" returns 19.96
only <scr_rigth_nexp> item = 010
<delim_sexp> = ""  and  <_oper_arg_sexp> = "_*._2.5" returns 0.025 % If left
side is "", 1 is used !*/!
no source lists, <start_nexp> = 5, <end_nexp> = 6,
<delim_sexp> = ""  and  <_oper_arg_sexp> = "_-i" returns  -5 and -6 % An index
list function
```

Warning: A source list can not be an output list in the same command.

Note: If using BigDecimal, the special Floating Point numbers NaN and Infinity are not supported.

Example of Creating an Index List:

```
List.create n, indexList
iStart = 1
iEnd = 10
List.join indexList, , , "", "_D_-i", iStart, iEnd    % Returns from -1 to -10
List.add  indexList, 0
List.join indexList, , , "", "_D_+i", iStart, iEnd, 1 % Adds from +1 to +10
List.sort indexList, 0                                % Sort mode ascending
List.row.print indexList, 1      % Shows 21 items from -10 to 10 in the console.
```

35.13 List.join.2d

Syntax: List.join.2d <xyList_nexp>, <xSource_nexp>, <ySource_nexp>{, <add_nexp>}

Joins the numeric x and y source Lists <**x**Source_nexp> and <**y**Source_nexp> into the returned **xy** List <**xy**List_nexp>. If the optional <add_nexp> is greater than 0, the result will be added into the returned list, otherwise the list will be cleared first. Default is 0.

Note, all Lists must be numeric.

Example:

```
List.join.2D xyPoints, xPoints, yPoints
```

35.14 List.join.3d

Syntax: List.join.3d <xyList_nexp>, <x(y)Source_nexp>, {<ySource_nexp>}, {<zSource_nexp>} {{, <add_nexp>}, <preZ_nexp>}

Joins the numeric x, y and z source Lists <**x**(y)Source_nexp> and <**y**Source_nexp>, <**z**Source_nexp> into the returned **xyz** List <**xyz**List_nexp>. If the optional <add_nexp> is greater than 0, the result will be added to the returned list, otherwise the list will be cleared first. Default is 0.

If the optional <**y**Source_nexp> is not given, <**x**(**y**)Source_nexp> is used as a **xy** list.

If the optional <**z**Source_nexp> is not given, predefined **z** values by the optional <preZ_nexp> are inserted. If <preZ_nexp> not given, the default value of 0.0 is inserted.

Note, all lists must be numeric.

Example 1:

```
List.join.3D xyzPoints, xPoints, yPoints, zPoints
```

Example 2:

```
List.join.3D xyzPoints, xyPoints, , zPoints
```

Example 3:

```
List.join.3D xyzPoints, xPoints, yPoints, , 0, preZ
```

Example 4:

```
List.join.3D xyzPoints, xyPoints, , , 0, preZ
```

35.15 List.kill.last

Syntax: List.kill.last

Kills the last list of the internal lists list. Typically, it is the last list that was created. Lists are global so if you create a list within a function, you are able to kill this list before leaving the function.

35.16 List.map.2d

Syntax: List.map.2d <pointer_nexp>, {{{{{{{{dx1}, dy1}, agl1}, dx2}, dy2}, agl2}, mulx}, muly}

All Arguments are of the type <nexp>.

Maps results of 2D operations by translation 1, rotation 1, translation 2, rotation 2 and multiplication (in this order) back into a given **x/y** value list.

This covers different cases of a combined workflow, which is often needed.

The arguments for translation are dx1, dy1, dx2 and dy2 (mostly in longitudinal units).

The arguments for rotation are agl1 and agl2 in degrees.

The arguments for multiplication are mulx and muly.

The default arguments for translation and rotation are set to 0.

The default arguments for multiplication are set to 1.

For a complete 2D operation you need the x and the y value, in this consequence an even number of values.

If the number of list items is not even, the last value is only computed by dx1, dx2 and mulx. So you can also use 1D value lists if the x/y arguments are equal and agl1 respectively agl2 are zero.

If you use **Screen Coordinates** you get a **clockwise** rotation.

If you use **World Coordinates** respectively the **Right-Hand Rule** you a get **counter-clockwise**

rotation or known as a mathematically positive rotation direction.

Example:

```
List.create n, l2d
List.add l2d,   0, 1,   1, 1,   7  %The number of list items is not even!
List.map.2d l2d, 0,0,  45,   0,0,   0,  2,2
Debug.on
Debug.dump.list l2d % Look what happens with the 7.
```

35.17 List.map.3d

Syntax: List.map.3d <pointer_nexp>, {{{{{{{{{{{{{dx1}, dy1}, dz1}, agl1x}, agl1y}, agl1z}, dx2}, dy2}, dz2}, agl2x}, agl2y}, agl2z}, mulx}, muly}, mulz}

All Arguments are of the type <nexp>.

Maps results of 3D operations by translation 1, rotation 1, translation 2, rotation 2 and multiplication (in this order) back into a given **x/y/z** value list.

This covers different cases of a combined workflow, which is often needed.

The arguments for translation are dx1, dy1,dz1, dx2, dy2 and dz2 (mostly in longitudinal units).

The arguments for rotation are agl1x, agl1y, agl1z, agl2x , agl2y and agl2z in degrees.

The arguments for multiplication are mulx, muly and mulz.

The default arguments for translation and rotation are set to 0.

The default arguments for multiplication are set to 1.

For a complete 3D operation you need the x, y and the z value, in this consequence the division from the number of list items by 3 has to be an integer.

If the list has one or two items more, this items are in **opposite** to List.map.2d **not changed**.

If you use **Screen Coordinates** you get a **clockwise** rotation.

If you use **World Coordinates** respectively the **Right-Hand Rule** you a **get counter-clockwise** rotation or known as a mathematically positive rotation direction.

Example:

```
List.create n, l3d
List.add l3d,  0, 1, 0,  1, 1, 0, 7 % The division of the number of list items
                                    % by 3 is not an integer.
Array.load t3d1[], 0,0,0            % Translation 1  [x,y,z]
Array.load r3a1[], 0,0,45           % Rotation 1 in degrees around [x,y,z]-Axis
Array.load t3d2[], 0,0,0            % Translation 2  [x,y,z]
Array.load r3a2[], 0,0,0            % Rotation 2 in degrees around [x,y,z]-Axis
Array.load m3d[], 1,1,1             % Multiplication  [x,y,z]
List.map.3d l3d, t3d1[1],t3d1[2], t3d1[3], r3a1[1], r3a1[2], r3a1[3], ~
 t3d2[1],t3d2[2], t3d2[3], r3a2[1], r3a2[2], r3a2[3], m3d[1], m3d[2], m3d[3]
Debug.on
Debug.dump.list l3d                 % Look what happens with the 7.
```

35.18 List.match

Syntax: List.match {<index_nexp>}, {<result_nexp>}, <source_nexp>, <by_find_sexp> {{{{{, <start_nexp>}, <end_nexp>}, <add_nexp>}, <mode_sexp>}, <inverse_sexp>}

Checks the expression <by_find_sexp> for matches in the source list <source_nexp>. Then puts the corresponding index into the optional list <index_nexp>, and the result into the optional <result_nexp>.

The type of the source list (S or N) is automatically detected to set the type of the output <result_nexp>.

The parameters <start_nexp> and <end_nexp> limit the search to the specified area of the source list. If no value is given, the entire list is searched.

If <add_nexp> is set to 1, the result will be added to the output list. Otherwise, the output list will first be cleared.

Options of matching mode <mode_sexp> are ("_RegEx" stands for regular expressions):

String expressions	Numeric expressions
_Default = _Is_In	_Default = _Is_In_IgnoreCase
_Is_In_IgnoreCase	_Starts_With_IgnoreCase
_Starts_With	_Ends_With_IgnoreCase
_Starts_With_IgnoreCase	_Equals_IgnoreCase
_Ends_With	_RegEx_First_IgnoreCase
_Ends_With_IgnoreCase	_=
_Equals_Equals_IgnoreCase	_<
_RegEx_Not	_<=
_RegEx_First	_>
_RegEx_First_IgnoreCase.	_>=
	_Equals_Numeric

Options for the inverse mode <inverse_sexp> are:

<inverse_sexp>	Description
_!	Opposite of the matches will be returned.
_Not	Opposite of the matches will be returned.
empty string	Default

Note: The source list can not be the output list at the same time.

35.19 List.remove

Syntax: List.remove <pointer_nexp>,<index_nexp>{{, <start_nexp>}, <end_nexp>}

Removes the list element specified by <index_nexp> from the list pointed to by <pointer_nexp>. The index is ones based. The first element of the list is 1.

Optionally, a range from <start_nexp> to <end_nexp> can be removed.

35.20 List.replace

Syntax: List.replace <pointer_nexp>, <index_nexp>, <sexp>|<nexp>{, <index_nexp>, <sexp>|

<nexp>}...

The list element specified by <index_nexp> in the list pointed to by <pointer_nexp> is replaced by the value of the string or numeric expression. The index is one-based. The first element of the list is 1.

The replacement expression type (string or numeric) must match the list type. Several elements can be replaced at the same time.

35.21 List.row.print

Syntax: List.row.print <pointer_nexp>{{{, <step_nexp>}, <lineNum_nexp>}, <result_svar>}

Prints a list as rows to the console or a string. The list step range is defined by <step_nexp>, which returns the row. Default is 1. Entries at the end of the list are returned even if they do not exactly match the step range. If <lineNum_nexp> is grater than 0, line numbers are returned at the start of the row. Default is 1. <result_svar> will optionally transfer ,the rows to a string with line feed endings ("\n"). In this case no console output is returned.

Example:

```
List.create n, xyzPoints
List.add xyzPoints,  0,0,0,  250,100,0,  500,0,0,  500,500,0, 250,400,0,
0,500,0
List.row.print  xyzPoints, 3 % Returns rows with line numbers and x, y, z values
```

35.22 List.search

Syntax: List.search <pointer_nexp>, value|value$, <result_nvar>{,<start_nexp>}

Searches the specified list for the specified string or numeric value. The position of the first occurrence is returned in the numeric variable <result_nvar>. If the value is not found in the list then the result is zero.

If the optional start expression parameter is present, the search starts at the specified element. The default start position is 1.

35.23 List.size

Syntax: List.size <pointer_nexp>, <nvar>

The size of the list pointed to by the list pointer is returned in the numeric variable <nvar>.

35.24 List.sort

Syntax: List.sort <pointer_nexp>{{{, <sort_mode_nexp>}, <locale_sexp>}, <strength_sexp>}

Sorts the content of the given list.

Options: <sort_mode_nexp>:

- 0 Sorted in ascending, UTF-8 character table numbered order (default)
- 1 Sorted in descending, UTF-8 character table numbered order

If the <locale_sexp> is set, the output is based on language and region. The locale specifies the language and region with standardized codes. The <locale_sexp> is a string containing zero or more codes separated by underscores.

The function accepts up to three codes. The first must be a language code, such as "en", "de" or "ja".

The second must be a region or country code, such as "FR", "US", or "IN". Some language and country combinations can accept a third code, called the "variant code".

The function also accepts the standard three-letter codes and numeric codes for country or region. For example, "fr_FR", "fr_FRA", and "fr_250" are all equivalent.

If <locale_sexp> = "", the default locale is used.

If <locale_sexp> is not given, the list is sorted by the order of the character map like **Array.sort**.

To control the strength of the sorting use <strength_sexp> with the keys described in **List.sort.by**.

35.25 List.sort.by

Syntax: List.sort.by {<index_nexp>}, {<toSort_nexp>}, <by_nexp>{{{, <sort_mode_nexp>}, <locale_sexp>}, <strength_sexp>}

Creates an optional index list <index_nexp> by sorting the content of the given list <by_nexp>. Optional will be the list <toSort_nexp> ordered by the internally created index.

The list <index_nexp> will be overwriten.

The lists <toSort_nexp> and <by_nexp> should be the same size and their types numeric and string can be combined.

Options: <sort_mode_nexp>:

- 0 Sorted in ascending, UTF-8 character table numbered order (default)
- 1 Sorted in descending, UTF-8 character table numbered order

If the <locale_sexp> is set, the output is based on language and region. The locale specifies the language and region with standardized codes. The <locale_sexp> is a string containing zero or more codes separated by underscores.

The function accepts up to three codes. The first must be a language code, such as "en", "de" or "ja".

The second must be a region or country code, such as "FR", "US", or "IN". Some language and country combinations can accept a third code, called the "variant code".

The function also accepts the standard three-letter codes and numeric codes for country or region. For example, "fr_FR", "fr_FRA", and "fr_250" are all equivalent.

If <locale_sexp> = "" , meaning "use my default locale".

If <locale_sexp> = empty, the list is sorted by the order of the character map like Array.sort.

To control the strength of the sorting use <strength_sexp> with the following keys.
"_Primary". Typically, recognizes differences in the base character so that "a" is smaller than "b". There are no differences between accents and umlauts, so that "a", "ä" and "á" are the same.

"_Secondary". **Is the default key**. Detects characters with accents. So "a" and "á" are not the same anymore as in _Primary. Accents in the characters are considered secondary differences (for example, "as" < "às" < "at"). Other differences between letters can also be considered secondary differences, depending on the language. A secondary difference is ignored when there is a primary difference anywhere in the strings.

"_Tertiary". Distinguishes in upper and lower case; in _Primary and _Secondary the spelling does not

matter, and "a" is equal to "A". Upper and lower case differences in characters are distinguished at tertiary strength (for example, "ao" < "Ao" < "aò"). In addition, a variant of a letter differs from the base form on the tertiary strength (such as "A" and "Ⓐ"). Another example is the difference between large and small Kana. A tertiary difference is ignored when there is a primary or secondary difference anywhere in the strings.

"_Identical". Really all Unicode characters are different. While the first three constants treat non-visible characters like CHR$(1) or CHR$(6) the same, they're really different under _Identical. When all other strengths are equal, the _Identical strength is used as a tiebreaker. For example, Hebrew cantillation marks are only distinguished at this strength. This strength should be used sparingly, as only code point value differences between two strings are an extremely rare occurrence. Using this strength substantially decreases the performance.

Examples for the "de" language code:

_Primary

abc = ABC

Quäken = Quaken

boß = boss

boß < boxen

_Secondary

abc = ABC

Quäken > Quaken

boß = boss

boß < boxen

_Tertiary

abc < ABC

Quäken > Quaken

boß > boss

boß < boxen

Example for special German sorting:

```
Debug.on
List.create s, toSort, by
List.add toSort, "Goldmann", "Göbel", "Goethe", "Göthe", "Götz"
Print "List of German words to sort"
Debug.dump.list toSort
List.toArray toSort, mem$[] % Copy the content of the toSort list
List.add.array by, mem$[] % into the by list.
Print "German DIN 5007 variant 1; used for words, such as in dictionaries"
List.sort.by  , toSort, by,  , "de", "_Secondary"
Debug.dump.list toSort
List.size by, ls
```

```
Print "German DIN 5007 variant 2; special sorting for name lists, ";
Print "for example in telephone directories"
For i = 1 To lS
 List.get by, i, str$
 str$ = Lower$(str$)
 str$ = Replace$(str$, "ä", "ae")
 str$ = Replace$(str$, "ö", "oe")
 str$ = Replace$(str$, "ü", "ue")
 List.replace by, i, str$
Next
List.clear toSort % Refresh the toSort list
List.add.array toSort, mem$[] % with the original content.
List.sort.by  , toSort, by,  , "de", "_Secondary"
Debug.dump.list toSort

Print "Austrian telephone directory sorting"
List.clear by % Refresh the toSort list
List.add.array by, mem$[] % with the original content.
List.size by, lS
For i = 1 To lS
 List.get by, i, str$
 str$ = Lower$(str$)
 str$ = Replace$(str$, "ä", "az")
 str$ = Replace$(str$, "ö", "oz")
 str$ = Replace$(str$, "ü", "uz")
 str$ = Replace$(str$, "ß", "ssz")
 str$ = Replace$(str$, "st.", "sankt")
 List.replace by, i, str$
Next
List.clear toSort
List.add.array toSort, mem$[]
List.sort.by  , toSort, by,  , "de", "_Secondary"
Debug.dump.list toSort
```

Example how to use a sorting index:

```
Array.load x[], 10, 2, 9, 7, -8, 99, 4, 7,1 , 4, -13
Array.load y[], -8, 99, 4, 7, 1, 4, -13, 10, 2, 9, 7
Array.load z[], 10, 2, 9, 1, 4, -13, 7, -8, 99, 4, 7
Print "In this case the source data have not to be copied or changed."
Print " X", "   Y", "   Z", " sorted by Y"
List.create n, byY, idxList
List.add.array byY, y[]
List.sort.by idxList, , byY
List.toArray idxList, idx[]
Array.length aL, idx[]
For i = 1 To aL
 If i = 1 Then Print "The min: ";
 If i = aL Then Print "The max: ";
 Print x[idx[i]], y[idx[i]], z[idx[i]]
Next
```

35.26 List.split

Syntax: List.split {<left_nexp>}, {<right_nexp>}, <source_nexp>, <by_reg_sexp>
{{{, <start_nexp>}, <end_nexp>}, <add_nexp>}

Splits the source list <source_nexp> into two lists and places them into <left_nexp> and <right_nexp> by the regular expression <by_reg_sexp>, item by item. If the right part does not exist, an empty string "" or 0.0 is returned.

With the list type (S or N) you control the type of your output. The type of the source list is detected automatically.

The <left_nexp> and <right_nexp> lists are optional, but you definitely need one.

The parameters <start_nexp> and <end_nexp> point to the range of the source list to work with. If no value is given, the whole list is used.

If the <add_nexp> argument is 1, the results will be added (appended) to the output lists. Else, the output lists are cleared before execution.

Note: The source list must not be an output list.

Example:

```
Array.load in1[], 1,2,27,4,5,6,7,8,3,4,57,114,115
List.create s, resLeft
List.create n, source
List.add.array source, in1[]
! resLeft will be contain a list of numbers as Strings
List.split resLeft, , source, "dummy"
Debug.on
Debug.dump.list resLeft
! And backwards again
List.split source, , resLeft, "dummy"
Debug.dump.list source

! If you deal with BigDecimal numbers, this makes also sense as
! a faster solution in opposite to BigD.int and BibD.frac.
List.create s, resRight
! The point needs in Regular Expressions a double backslash as a special char.
List.split resLeft, resRight, source, "\\."
Print "Int:"
Debug.dump.list resLeft
Print "Frac:"
Debug.dump.list resRight
```

35.27 List.split.2d

Syntax: List.split.2d <xList_nexp>, <yList_nexp>, <xySource_nexp>{, <add_nexp>}

Splits a numeric **xy** source list <**xy**Source_nexp> into the returned x and y lists. If the optional <add_nexp> is greater than 0, the results will be added (appended) to the returned lists. Otherwise the lists will first be cleared. Default is 0.

Note: All lists have to be numeric.

Example:

```
List.split.2D xPoints, yPoints, xyPoints
```

35.28 List.split.3d

Syntax: List.split.3d <x(y)List_nexp>, {<yList_nexp>}, {<zList_nexp>}, <xyzSource_nexp>{, <add_nexp>}

Splits a numeric **xyz** source List <**xyz**Source_nexp> into the returned **x**, **y** and **z** Lists. If the optional <add_nexp> is greater than 0, the results will be added (appended) to the returned lists. Otherwise, the lists will first be cleared. Default is 0.

If the optional <**y**List_nexp> is not given, <**x**(**y**)List_nexp> returns a **xy** List.

Note: All lists must be numeric.

Example 1:

```
List.split.3D xPoints, yPoints, zPoints, xyzPoints
```

Example 2:

```
List.split.3D xPoints, yPoints, , xyzPoints
```

Example 3:

```
List.split.3D xyPoints, , zPoints, xyzPoints
```

Example 4:

```
List.split.3D xyPoints, , , xyzPoints
```

35.29 List.spread

Syntax: List.spread <listOfLists_pointer_nexp>, Array[{<start>, <length>}], <count_nexp>{, <clear_nexp>}

or

Syntax: List.spread <listOfLists_pointer_nexp>, Array$[{<start>, <length>}], <count_nexp>{, <clear_nexp>}

<listOfLists_pointer_nexp> must point to an existing list whose items are themselves pointers to lists. This command spreads <count_nexp> number of elements, section by section, from a given Array range into lists defined by the ist pointers in list <listOfLists_pointer_nexp>. If optional <clear_nexp> is set to 1, the lists will first be cleared. The lists and the array must be of the same type.

35.30 List.toArray

Syntax: List.toArray <pointer_nexp>, Array$[] | Array[]

Copies the list pointed to by the list pointer into an array. The array type (string or numeric) must be the same as the list type. If the array exists, it is overwritten, otherwise a new array is created. The result is always a one-dimensional array.

35.31 List.type

Syntax: List.type <pointer_nexp>, <svar>

The type of list pointed to by the list pointer is returned in the string variable <svar>.

- Returns the upper case character "S" if the list is a list of strings.
- Returns the upper case character "N" if the list is a list of numbers.

36 Math Functions

Math functions act like numeric variables in a <nexp> (or <lexp>).

Numeric expressions are evaluated step-by-step, following the BODMAS (Bracket, Of, Divide, Multiply, Add, Subtract) rule. This rule goes by different names in different countries. In the USA it is usually called PEMDAS (Parentheses, Exponents, Multiplication/Division, Addition/Subtraction). For more information see https://en.wikipedia.org/wiki/Order_of_operations.

For example, 8 ÷ 2 × (2 + 2) is equivalent to 8 ÷ 2 × 4, also equivalent to 4 × 4 which is equal to 16. Therefore:

```
Print 8 / 2 * (2 + 2)  % yields 16
```

Another example, 8 ÷ 2 (2 + 2) is equivalent to 8 ÷ 2 (4), also equivalent to 8 ÷ 8 which is equal to 1. Therefore:

```
Print 8 / (2 * (2 + 2))  % yields 1
```

Happily and notably, **And** (**&**) and **Or** (**|**) operators are at the top of the hierarchy, so you can conjoin/disjoin booleans without additional parentheses:

```
If a > 5 & a < 10 Then ...
```

Same with the **Not** (**!**) operator:

```
a = 3
Print !a < 4     % yields FALSE
Print !(a < 4)   % yields FALSE
Print (!a) < 4   % yields TRUE
```

See the chapter on Numbers for a description on the Double type.

36.1 Abs

Syntax: Abs(<nexp>)

Returns the absolute value of <nexp>.

36.2 Acos

Syntax: Acos(<nexp>)

Returns the arc cosine of the angle <nexp>, in the range of 0.0 through pi.The units of the angle are radians. If the value of <nexp> is less than -1 or greater than 1, the function generates a runtime error.

36.3 Asin

Syntax: Asin(<nexp>)

Returns the arc sine of the angle <nexp>, in the range of -pi/2 through pi/2. The units of the angle are radians. If the value of <nexp> is less than -1 or greater than 1, the function generates a runtime error.

36.4 Atan

Syntax: Atan(<nexp>)

Returns the arc tangent of the angle <nexp>, in the range of -pi/2 through pi/2. The units of the angle are radians.

Special cases:

- Atan(+0.0) = +0.0
- Atan(-0.0) = -0.0
- Atan(+infinity) = +pi/2
- Atan(-infinity) = -pi/2
- Atan(NaN) = NaN

36.5 Atan2

Syntax: Atan2(<nexp_y>, <nexp_x>)

Returns the angle *theta* from the conversion of rectangular coordinates (x, y) to polar coordinates (r,*theta*). Please note the order of the parameters in this function.

Special cases:

- Atan2((anything), NaN) = NaN;
- Atan2(NaN , (anything)) = NaN;
- Atan2(+0.0, +(anything but NaN)) = +0.0
- Atan2(-0.0, +(anything but NaN)) = -0.0
- Atan2(+0.0, -(anything but NaN)) = +pi
- Atan2(-0.0, -(anything but NaN)) = -pi
- Atan2(+(anything but 0 and NaN), 0) = +pi/2
- Atan2(-(anything but 0 and NaN), 0) = -pi/2
- Atan2(+(anything but infinity and NaN), +infinity) = +0.0
- Atan2(-(anything but infinity and NaN), +infinity) = -0.0
- Atan2(+(anything but infinity and NaN), -infinity) = +pi
- Atan2(-(anything but infinity and NaN), -infinity) = -pi
- Atan2(+infinity, +infinity) = +pi/4 **+ 45° 1. Quadrant**
- Atan2(-infinity, +infinity) = -pi/4 **- 45° 4. Quadrant**
- Atan2(+infinity, -infinity) = +3pi/4 **+ 135° 2. Quadrant**
- Atan2(-infinity, -infinity) = -3pi/4 **- 135° 3. Quadrant**
- Atan2(+infinity, (anything but,0, NaN, and infinity)) = +pi/2
- Atan2(-infinity, (anything but,0, NaN, and infinity)) = -pi/2

Example:

```
Fn.def ATAN4(y, x) % Returns the radiant for y / x in the range [0 … 2*PI]
 result = Atan2(y, x)
 If result < 0 Then result = Pi()*2 + result
 If result = PI()*2 Then result = 0
 Fn.rtn result
Fn.end

Print ATAN4(-1, -1), ToDegrees(ATAN4(-1, -1))
```

36.6 BAnd

Syntax: BAnd(<nexp1>, <nexp2>)

Returns the logical bitwise value of <nexp1> AND <nexp2>. The double-precision floating-point values are converted to 64-bit integers before the operation.

```
Band(3,1) is 1
```

36.7 BNot

Syntax: BNot(<nexp>)

Returns the bitwise complement value of <nexp>. The double-precision floating-point value is converted to a 64-bit integer before the operation.

```
Bnot(7) is -8
Hex$(Bnot(Hex("1234"))) is fffffffffffedcb
```

36.8 BOr

Syntax: BOr(<nexp1>, <nexp2>)

Returns the logical bitwise value of <nexp1> OR <nexp2>. The double-precision floating-point values are converted to 64-bit integers before the operation.

```
Bor(1,2) is 3
```

36.9 BXor

Syntax: BXor(<nexp1>, <nexp2>)

Returns the logical bitwise value of <nexp1> XOR <nexp2>. The double-precision floating-point values are converted to 64-bit integers before the operation.

```
Bxor(7,1) is 6
```

36.10 Cbrt

Syntax: Cbrt(<nexp>)

Returns the closest double-precision floating-point approximation of the cube root of <nexp>.

36.11 Ceil

Syntax: Ceil(<nexp>)

Rounds up towards positive infinity. 3.X becomes 4 and -3.X becomes -3.

36.12 Clamp

Syntax: Clamp(<value_nexp>, <min_nexp>, <max_nexp>)

Ensures that the numerical value <value_nexp> fits in a given numerical range. If the number is smaller than the minimum <min_nexp> required by the range, then the minimum <min_nexp> of the range will be returned. If the number is higher than the maximum <max_nexp> allowed by the range then the maximum <max_nexp> of the range will be returned.

36.13 Cos

Syntax: Cos(<nexp>)

Returns the trigonometric cosine of angle <nexp>. The units of the angle are radians.

36.14 Cosh

Syntax: Cosh(<nexp>)

Returns the trigonometric hyperbolic cosine of angle <nexp>. The units of the angle are radians.

36.15 Even

Syntax: Even(<nexp>)

Returns 1 if the integer part of <nexp> is even. Otherwise 0 is returned. If <nexp> is NaN or infinite, 0 is also returned. You may have to first **Round** for cases like 79.999999999.

36.16 ExpXP

Syntax: ExpXP(<nexp>)

Returns e raised to the <nexp> power.

36.17 Floor

Syntax: Floor(<nexp>)

Rounds down towards negative infinity. 3.X becomes 3 and -3.X becomes -4.

36.18 Frac

Syntax: Frac(<nexp>)

Returns the fractional part of <nexp>. 3.4 becomes 0.4 and -3.4 becomes -0.4.

Although **Frac(n)** is four times slower than its equivalent, n – **Int(n)**, it is more accurate.

36.19 Hypot

Syntax: Hypot(<nexp_x>, <nexp_y)

Returns Sqr(x^2+y^2) without intermediate overflow or underflow.

36.20 Int

Syntax: Int(<nexp>)

Returns the integer part of <nexp>. 3.X becomes 3 and -3.X becomes -3. This operation may also be called truncation, rounding down, or rounding toward zero. In other words, the decimal point and subsequent digits are deleted.

Note: This function works differently in most other BASIC dialects. See **Round()** for more details.

36.21 Is_infinite

Syntax: Is_infinite(<nexp>)

Returns 1.0 (true) if <nexp> is an Infinity Floating Point number, else 0.0 if not.

36.22 Is_NaN

Syntax: Is_NaN(<nexp>)

Returns 1.0 (true) if <nexp> is NaN (Not a Number) Floating Point number, else 0.0 if not.

36.23 Log

Syntax: Log(<nexp>)

Returns the natural logarithm (base e) of <nexp>.

36.24 Log10

Syntax: Log10(<nexp>)

Returns the base 10 logarithm of the <nexp>.

36.25 Max

Syntax: Max(<nexp>, <nexp>)

Returns the maximum of two numbers as an <nvar>.

36.26 Min

Syntax: Min(<nexp>, <nexp>)

Returns the minimum of two numbers as an <nvar>.

36.27 Mod

Syntax: Mod(<nexp1>, <nexp2>)

Returns the remainder of <nexp1> divided by <nexp2>. If <nexp2> is 0, the function returns NaN.

36.28 Odd

Syntax: Odd(<nexp>)

Returns 1 if the integer part of <nexp> is odd. Otherwise 0 is returned. If <nexp> is NaN or infinite, 0 is also returned. You may have to first **Round** for cases like 80.999999999.

36.29 Pi

Syntax: Pi()

Returns the double-precision floating-point value closest to pi.

36.30 Pow

Syntax: Pow(<nexp1>, <nexp2>)

Returns <nexp1> raised to the <nexp2> power.

36.31 Round

Syntax: Round(<value_nexp>{, <scale_nexp>{, <roundingMode_sexp>}})

In it simplest form, **Round(<value_nexp>)**, this function returns the whole number closest to <nexp>. You can use the optional parameters to specify more complex operations.

The <scale_nexp> is an optional decimal place count. It sets the number of places to the right of the decimal point. The last digit is rounded. if <scale_mode> is < 0, only a faster Half-up rounding mode is used. Omitting the parameter is the same as setting it to zero.

<roundingMode_sexp> is an optional rounding mode. It is a one- or two-character mnemonic code that tells **Round()** what kind of rounding to do. It is not case-sensitive. There are eight rounding modes:

Mode:	Meaning:	-3.8	-3.5	-3.1	-3.0	3.0	3.1	3.5	3.8
"HD"	Half-down	-4.0	-3.0	-3.0	-3.0	3.0	3.0	3.0	4.0

"HE"	Half-even	-4.0	-4.0	-3.0	-3.0	3.0	3.0	4.0	4.0
"HU"	Half-up	-4.0	-4.0	-3.0	-3.0	3.0	3.0	4.0	4.0
"D"	Down	-3.0	-3.0	-3.0	-3.0	3.0	3.0	3.0	3.0
"U"	Up	-4.0	-4.0	-4.0	-3.0	3.0	4.0	4.0	4.0
"F"	Floor	-4.0	-4.0	-4.0	-3.0	3.0	3.0	3.0	3.0
"C"	Ceiling	-3.0	-3.0	-3.0	-3.0	3.0	4.0	4.0	4.0
"LI"	Legacy int()*	-4.0	-4.0	-4.0	-3.0	3.0	3.0	3.0	3.0

In this table, "down" means "toward zero" and "up" means "away from zero" (toward ±∞)

"Half" refers to behavior when a value is half-way between rounding up and rounding down(x.5 or -x.5). "Half-down" rounds x.5 towards zero and "half-up" rounds x.5 away from zero.

"Half-even" is either "half-down" or "half-up", whichever would make the result **even**. 4.5 and 3.5 both round to 4.0. "Half-even" is also called "banker's rounding", because it tends to average out rounding errors.

In most cases is **"Half-Up"→"HU"** the best choice.

***Legacy int() →"LI"** is compatible with most other BASIC dialects.

Examples of Most Other BASIC Dialects	Examples of BASIC!
`myNumber = Int(99.8) ' Returns 99`	`myNumber = round(99.8, 0, "LI") % Returns 99`
`myNumber = Fix(99.8) ' Returns 99`	`myNumber = INT(99.8) % Returns 99`
`myNumber = Int(-99.8)' Returns -100`	`myNumber = round(-99.8, 0, "LI")% Returns -100`
`myNumber = Fix(-99.8)' Returns -99`	`myNumber = INT(-99.8) % Returns -99`
`myNumber = Int(-99.2)' Returns -100`	`myNumber = round(-99.2, 0, "LI")% Returns -100`
`myNumber = Fix(-99.2)' Returns -99`	`myNumber = INT(-99.2) % Returns -99`

If you do not provide a <roundingMode_sexp>, **Round()** adds +0.5 and rounds down (toward zero). This is legacy behavior, copied from earlier versions of BASIC!. **Round(n)** is NOT the same as **Round(n, 0)**.

Round() generates a runtime error if <scale_nexp> < 0 or <roundingMode_sexp> are not valid.

Examples:

```
pi = Round(3.14159)              % pi is 3.0
pi = Round(3.14159, 2)           % pi is 3.14
pi = Round(3.14159, , "U")       % pi is 4.0
pi = Round(3.14159, 4, "F")      % pi is 3.1415
negpi = Round(-3.14159, 4, "D") % negpi is -3.1416
```

Note that **Floor(n)** is exactly the same as **Round(n, 0, "F")**, but **Floor(n)** is a little faster. In the same way, **Ceil(n)** is the same as **Round(n, 0, "C")**, and **Int(n)** is the same as **Round(n, 0, "D")**.

36.32 Sgn

Syntax: Sgn(<nexp>)

Returns the signum function of the numerical value of <nexp>, representing its sign.

When the value is:	Return:
> 0	1
= 0	0
< 0	-1

36.33 Shift

Syntax: Shift(<value_nexp>, <bits_nexp>)

Shifts the value <value_nexp> by the bit count <bits_nexp>. If the bit count is < 0, the value will be shifted left. If the bit count is > 0, the bits will be shifted right. The right shift will replicate the sign bit. The double-precision floating-point value are truncated to 64-bit integers before the operation.

36.34 Sin

Syntax: Sin(<nexp>)

Returns the trigonometric sine of angle <nexp>. The units of the angle are radians.

36.35 Sinh

Syntax: Sinh(<nexp>)

Returns the trigonometric hyperbolic sine of angle <nexp>. The units of the angle are radians.

36.36 Sqr

Syntax: Sqr(<nexp>)

Returns the closest double-precision floating-point approximation of the positive square root of <nexp>. If the value of <nexp> is negative, the function generates a runtime error.

36.37 Tan

Syntax: Tan(<nexp>)

Returns the trigonometric tangent of angle <nexp>. The units of the angle are radians.

36.38 ToDegrees

Syntax: ToDegrees(<nexp>)

Converts <nexp> angle measured in radians to an angle measured in degrees.

36.39 ToRadians

Syntax: ToRadians(<nexp>)

Converts <nexp> angle measured in degrees to an angle measured in radians.

37 Math Functions, Big Decimal

See the chapter on Numbers for a description on the BigDecimal type.

37.1 BigD.abs

Syntax: BigD.abs <result_svar>, <first_sexp>

<result_svar> is set to a BigDecimal string whose value is the absolute value of <first_sexp>. The scale of the result is the same as the scale of <first_sexp>.

37.2 BigD.add

Syntax: BigD.add <result_svar>, <first_sexp>, <second_sexp>

<result_svar> is set to a BigDecimal string whose value is <first_sexp> + <second_sexp>. The scale of the result is the maximum of the scales of the two arguments.

37.3 BigD.compare

Syntax: BigD.compare <result_nvar>, <first_sexp>, <second_sexp>

Compares BigDecimal <first_sexp> with <second_sexp> and sets <result_svar> to one of the three values 1, 0, or -1. The method behaves as if <first_sexp> - <second_sexp> is computed. If the difference is > 0 then 1 is returned, if the difference is < 0 then -1 is returned, and if the difference is 0 then 0 is returned. This means, that if two decimal instances are compared which are equal in value but differ in scale, then these two instances are considered as equal.

37.4 BigD.date

Syntax: BigD.date <result_svar>, <first_sexp>

Returns the time in milliseconds since January 1, 1970 00:00:00.0 UTC.

Example:

```
mDate$ = "2020-10-15T09:27:37Z+0100"
BigD.date result$, mDate$
```

37.5 BigD.divide

Syntax: BigD.divide <result_svar>, <first_sexp>, <second_sexp>, <scale_nexp>, <roundingMode_sexp>

<result_svar> is set to a BigDecimal string whose value is <first_sexp> / divisor <second_sexp>. The scale of the result set by <scale_nexp>. If rounding is required to meet the specified scale, the specified rounding mode <roundingMode_sexp> is applied.

See **BigD.round** for rounding details.

37.6 BigD.equals

Syntax: BigD.equals <result_nvar>, <first_sexp>, <second_sexp>

<result_svar> is set to 1.0 if <first_sexp> and <second_sexp> are BigDecimal instances and equal. Otherwise <result_svar> is set to 0.0. Two big decimals are equal if their unscaled value **and** their scale is equal. For example, 1.0 (10*10-1) is not equal to 1.00 (100*10-2). Similarly, zero instances are not equal if their scale differs.

37.7 BigD.frac

Syntax: BigD.frac <result_svar>, <first_sexp>

<result_svar> is set to a BigDecimal string whose value is the fractional part of <first_sexp>. The scale of the result is the same as the scale of <first_sexp>.

37.8 BigD.fromBase

Syntax: BigD.fromBase <result_svar>, <string_sexp>, <base_sexp>

<result_svar> is set to a BigDecimal string whose value is the result of the string <string_sexp> decoded by the base "_Bin", "_Oct" or "_Hex". Hex is the default, which is also used if <base_sexp> contains an illegal value.

37.9 BigD.fromDouble

Syntax: BigD.fromDouble <result_svar>, <number_nexp>

<result_svar> is set to a BigDecimal string from the Double <number_nexp>.

NaN (Not a Number), Double.POSITIVE_INFINITY or Double.NEGATIVE_INFINITY are not supported and throw a runtime error.

37.10 BigD.hashCode

Syntax: BigD.hashCode <result_svar>, <first_sexp>

<result_svar> is set to a BigDecimal string whose value is the hash code for <first_sexp>.

The hash code is computed as

s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]

using integer arithmetic, where s[i] is the ith character of the string, n is the length of the string, and ^ indicates exponentiation. The hash value of an empty string is zero.

37.11 BigD.int

Syntax: BigD.int <result_svar>, <first_sexp>

<result_svar> is set to a BigDecimal string whose value is the integral part of <first_sexp>.

37.12 BigD.max

Syntax: BigD.max <result_svar>, <first_sexp>, <second_sexp>

<result_svar> is set to a BigDecimal string whose value is the maximum of <first_sexp> and <second_sexp>.

37.13 BigD.min

Syntax: BigD.min <result_svar>, <first_sexp>, <second_sexp>

<result_svar> is set to a BigDecimal string whose value is the minimum of <first_sexp> and <second_sexp>.

37.14 BigD.movePointLeft

Syntax: BigD.movePointLeft <result_svar>, <first_sexp>, <n_nexp>

<result_svar> is set to a BigDecimal string whose value is <first_sexp> with its decimal point moved <n_nexp> places to the left. If <n_nexp> is less than 0, then the decimal point is moved to the right.

The result is obtained by changing the scale. If the scale of the result becomes negative, then its precision is increased such that the scale is zero.

Note that if <n_nexp> = 0 the result is mathematically equivalent, but the scale is ≥ 0.

37.15 BigD.movePointRight

Syntax: BigD.movePointRight <result_svar>, <first_sexp>, <n_nexp>

<result_svar> is set to a BigDecimal string whose value is <first_sexp> with its decimal point moved <n_nexp> places to the right. If <n_nexp> is less than 0, then the decimal point is moved to the left.

The result is obtained by changing the scale. If the scale of the result becomes negative, then its precision is increased such that the scale is zero.

Note that if <n_nexp> = 0 the result is mathematically equivalent, but the scale is ≥ 0.

37.16 BigD.multiply

Syntax: BigD.multiply <result_svar>, <first_sexp>, <second_sexp>

<result_svar> is set to a BigDecimal string whose value is <first_sexp> * <second_sexp>. The scale of the result is the sum of the scales of the two arguments.

37.17 BigD.nanoTime

Syntax: BigD.nanoTime <result_svar>

<result_svar> is set to a BigDecimal string whose value is the current timestamp of the most precise timer available on the local system, in nanoseconds. Equivalent to Linux's CLOCK_MONOTONIC.

This timestamp should only be used to measure a duration by comparing it against another timestamp on the same device. Values returned by this method do not have a defined correspondence to wall clock times; the zero value is typically whenever the device last booted. Use **BigD.time** if you want to know what time it is.

To compare two nanoTime values:

```
BigD.nanoTime  tic1$
...
BigD.nanoTime  tic2$
```

You should use **BigD.compare result, tic2$, tic1$**, because of the possibility of numerical overflow.

37.18 BigD.pow

Syntax: BigD.pow <result_svar>, <first_sexp>, <n_nexp>

<result_svar> is set to a BigDecimal string whose value is <first_sexp> raised to the <n_nexp> power. The scale of the result is n * the scale of <first_sexp>.

Note: **BigD.pow x$, 0.0, r$** returns "1", even if x$ = "0".

Note: The implementation is based on the ANSI standard X3.274-1996 algorithm.

Note: <n_nexp> must be between 0 and 999999999.

37.19 BigD.precision

Syntax: BigD.precision <result_nvar>, <first_sexp>

<result_svar> is set to a BigDecimal string whose value is the precision of <first_sexp>. The precision is the number of decimal digits used to represent this decimal. It is equivalent to the number of digits of the unscaled value. The precision of 0 is 1 (independent of the scale).

37.20 BigD.remainDividing

Syntax: BigD.remainDividing <integral_svar>, <remainder_svar>, <first_sexp>, <second_sexp>

<integral_svar> is set to a BigDecimal string which contains the integral part of <first_sexp> / divisor <second_sexp>, and <remainder_svar> is set to a BigDecimal string which contains the remainder <first_sexp> - <first_sexp>/int(divisor) * divisor.

37.21 BigD.round

Syntax: BigD.round <result_svar>, <first_sexp>, <scale_nexp>, <roundingMode_sexp>

<integral_svar> is set to a BigDecimal string whose value is <first_sexp>, rounded according to the scale in <scale_nexp> and rounding mode in <roundingMode_sexp>. If rounding is required to meet the specified scale, then the specified rounding mode <roundingMode_nexp> is applied.

There are eight rounding modes:

Mode:	Meaning:	-3.8	-3.5	-3.1	-3.0	3.0	3.1	3.5	3.8
"HD"	Half-down	-4.0	-3.0	-3.0	-3.0	3.0	3.0	3.0	4.0
"HE"	Half-even	-4.0	-4.0	-3.0	-3.0	3.0	3.0	4.0	4.0
"HU"	Half-up	-4.0	-4.0	-3.0	-3.0	3.0	3.0	4.0	4.0
"D"	Down	-3.0	-3.0	-3.0	-3.0	3.0	3.0	3.0	3.0
"U"	Up	-4.0	-4.0	-4.0	-3.0	3.0	4.0	4.0	4.0
"F"	Floor	-4.0	-4.0	-4.0	-3.0	3.0	3.0	3.0	3.0
"C"	Ceiling	-3.0	-3.0	-3.0	-3.0	3.0	4.0	4.0	4.0
"LI"	Legacy int()	-4.0	-4.0	-4.0	-3.0	3.0	3.0	3.0	3.0

In this table, "down" means "toward zero" and "up" means "away from zero" (toward ±∞)

In most cases is "**Half-Up**"→"**HU**" is the best choice.

37.22 BigD.scale

Syntax: BigD.scale <result_nvar>, <first_sexp>

<result_svar> is set to a Double whose value is the scale of BigDecimal <first_sexp>. The scale is the number of digits after the decimal point. The value of <first_sexp> is the unsigned Value * 10^{-scale}. If the scale is negative, then <first_sexp> represents a big integer.

37.23 BigD.sign

Syntax: BigD.sign <result_nvar>, <first_sexp>

<result_svar> is set to a Double whose value is the signum function of the BigDecimal value of <first_sexp>, representing its sign:

if BigDecimal of <first_sexp> < 0, <result_svar> is set to -1
if BigDecimal of <first_sexp> = 0, <result_svar> is set to 0
if BigDecimal of <first_sexp> > 0, <result_svar> is set to 1

37.24 BigD.sqr

Syntax: BigD.sqr <result_svar>, <first_sexp>, <scale_nexp>

<result_svar> is set to a BigDecimal string whose value is the closest approximation of the positive square root of <first_sexp>. If the value of <first_sexp> is negative, the function generates a runtime error. The maximum scale is given by <scale_nexp>.

37.25 BigD.subtract

Syntax: BigD.subtract <result_svar>, <first_sexp>, <second_sexp>

<result_svar> is set to a BigDecimal string whose value is <first_sexp> - <second_sexp>. The scale of the result is the maximum of the scales of the two arguments.

37.26 BigD.sum

Syntax: BigD.sum <result_svar>, Array$[]

<result_svar> is set to a BigDecimal string whose value is the sum of all string array items. The scale of the result is the maximum of the scales of all arguments.

37.27 BigD.time

Syntax: BigD.time <result_svar>

<result_svar> is set to a BigDecimal string whose value is the current time in milliseconds since January 1, 1970 00:00:00.0 UTC.

This method always returns UTC times, regardless of the system's time zone. This is often called "Unix time" or "epoch time".

This method shouldn't be used for measuring timeouts or other elapsed time measurements, as changing the system time can affect the results. Use BigD.nanoTime for that.

37.28 BigD.toBase

Syntax: BigD.toBase <result_svar>, <string_sexp>, <base_sexp>

<result_svar> is set to a BigDecimal string whose value is the result of the string <string_sexp> encoded by the base "_Bin", "_Oct" or "_Hex". Hex is the default, which is also used if <base_sexp> contains an illegal value.

37.29 BigD.toDouble

Syntax: BigD.toDouble <result_nvar>, <first_sexp>

<result_nvar> is set to the Double version of <first_sexp> BigDecimal. NaN (Not a Number), Double.POSITIVE_INFINITY or Double.NEGATIVE_INFINITY are not supported and throw a runtime error.

Note, that if the unscaled value has more than 53 significant digits, then this decimal cannot be represented exactly in a double variable. In this case the result is rounded.

37.30 BigD.toEngineering

Syntax: BigD.toEngineering <result_svar>, <first_sexp>

<result_nvar> is set to a string representation of <first_sexp> BigDecimal. This representation always has all significant digits of this value.

If the scale is negative or if scale - precision ≥ 6 then engineering notation is used. Engineering notation is similar to the scientific notation except that the exponent is made to be a multiple of 3 such that the integer part is ≥ 1 and < 1000.

37.31 BigD.toSientific

Syntax: BigD.toSientific <result_svar>, <first_sexp>

<result_nvar> is set to a canonical string representation of <first_sexp> BigDecimal. If necessary, scientific notation is used. This representation always has all significant digits of this value.

If the scale is negative or if scale - precision ≥ 6 then scientific notation is used.

37.32 BigD.ulp

Syntax: BigD.ulp <result_svar>, <first_sexp>

<result_nvar> is set to the unit in the last place (ULP) of <first_sexp> BigDecimal. A ULP is the distance to the nearest big decimal with the same precision.

The amount of a rounding error in the evaluation of a floating-point operation is often expressed in ULPs. An error of 1 ULP is often seen as a tolerable error.

For BigDecimal class, the ULP of a number is simply 10-scale. For example:

BigD.ulp “123”, r$ returns “1”

BigD.ulp “1.23”, r$ returns “0.01”

38 Menu Commands

38.1 MenuItem.get.datalink

Syntax: MenuItem.get.datalink <data_svar>

Returns the data of the last selected menu item in a human readable JSON string.

Example:

```
Begin: "{" +~
 "_MenuFrom:_Graphic" +~        % Other cases are "_Console" and "_HTML"
 "_TitleText:MyItemTitle" +~    % If is "_TitleUp" returned, the Home Button of the Title Bar is selected!
"_ItemId:20" +~                 % Returns the item id
"_GroupId:1" +~                 % Returns the group id
"_Checked:1"+~                  % Returns if checked 1 or not 0
"_Checkable:0"+~                % Returns if it is check-able 1 or not 0;
End:   "}"
```

39 Mesh Commands

39.1 Mesh.hull

Syntax: Mesh.hull <result_xyList_nexp>, <x(y)List_nexp>{, <yList_nexp>)}

Returns a xy List of a convex hull polygon around a 2D point cloud. The first list, defined by <x(y)List_nexp> can be contain x or xy values.

The optional y List defined by <yList_nexp> contain only y values. If the y list is not given, the first list has to be a xy list.

The result can be used directly by the **Gr.poly** command.

Example:

```
List.create n, re  % xy List
List.create n, pl
List.add re, -300, 200,100, 200,100, -400,-300, -400 , 400,500, 100,300
Mesh.hull pl, re
Gr.open "_LightGreen", 1,1 % Background
Gr.color "_Orange", 1 % Orgin
Gr.circle oPtr0, 500, 500, 20

Gr.color "_Red", 0 % The hull polygon
GR.poly oPtr2, pl, 500, 500

Gr.color "_Magenta", 1 % All points
List.size re, sz
For i = 1 To sz Step 2
 List.get re, i, x
 List.get re, i + 1, y
 Gr.circle oPtr, 500 + x, 500 + y, 10
Next
Gr.color "_Blue", 1 % Points along the hull polygon
List.size pl, sz
For i = 1 To sz Step 2
 List.get pl, i, x
 List.get pl, i + 1, y
 Gr.circle oPtr, 500 + x, 500 + y, 10
Next
Gr.render

Do
 Pause 100
Until 0
```

39.2 Mesh.stl.load

Syntax: Mesh.stl.load <triangles_xyzList_nexp>, <midpoints_xyzList_nexp>,
<normals_xyzList_nexp>, {<header_sval>}, <filePath_sexp>{, <normal_length_nexp>}

This command returns the xyz lists <triangles_xyzList_nexp>, <midpoints_xyzList_nexp> and <normals_xyzList_nexp> from the file named by <filePath_sexp>. The format is STL binary. The precision is 32 bit float. For more information see: https://en.wikipedia.org/wiki/STL_(file_format).

The optional <header_sval> returns the header of the binary STL file as an ASCII string in a length of 80 characters. If an error occurs the string of <header_sval> starts with "_Error". The absolute length of the plain normal vectors can be defined by <normal_length_nexp>. Default is no change.

If the file specifies more triangles than allowed by the file size, the command tries to load as many triangles as possible.

39.3 Mesh.stl.save

Syntax: Mesh.stl.save <triangles_xyzList_nexp>, <normals_xyzList_nexp>, <filePath_sexp>{{, <header_sexp>}, <normal_length_nexp>}

Saves the xyz Lists <triangles_xyzList_nexp> and <normals_xyzList_nexp> into the file named by <filePath_sexp>. The format is STL binary. The precision is 32 bit float. For more information, see https://en.wikipedia.org/wiki/STL_(file_format).

The optional <header_sexp> sets the header of the binary STL file as an ASCII string of up to 80 bytes. Longer ASCII strings will be cut off. Characters in the default UTF-8 format will be write as an ASCII character plus a question mark.

The absolute length of the plain normal vectors can be defined by <normal_length_nexp>. Default is no change.

39.4 Mesh.triangle

Syntax: Mesh.triangle <result_xyList_nexp>, <x(y)List_nexp>{, <yList_nexp>)}

Returns a xy list of triangle polygons within a 2D point cloud using the Delaunay triangulation. All triangle points are in clockwise order if you use screen coordinates.

Using the right hand order the points are sorted in the counterclockwise direction.

If you use **World Coordinates** respectively the **Right-Hand Rule,** also known as a mathematically positive, the points are sorted in the counterclockwise direction.

The first input list defined by <x(y)List_nexp> can be contain x or xy values. The optional y list defined by <yList_nexp> contain only y values. If the y list is not given, the first list has to be a xy list.

Make sure that you do not use points with equal xy coordinates, because this command does not check.

The result can be used directly by the **Gr.poly** command.

39.5 Mesh.triangle.midpoint

Syntax: Mesh.triangle.midpoint <result_xyList_nexp>, <x(y)List_nexp>{, <yList_nexp>)}

Returns a list of xy midpoints of triangle polygons within a 2D point cloud using the Delaunay triangulation.

The first input list defined by <x(y)List_nexp> can be contain x or xy values. The optional y list defined by <yList_nexp> contain only y values. If the y list is not given, the first list has to be a xy List.

Make sure that you do not use points with equal xy coordinates, because this command does not check.

The result can be used to refine (triple) the mesh by combining this result with the input points.

39.6 Mesh.triangle.2.5d

Syntax: Mesh.triangle.2.5d <xyzTriangles_nexp>, <xyzMidpoints_nexp>, <xyzNormals_nexp>, {<normal_length_nexp>}, <xyzSource_nexp>{{, <xyzOuterBorder_nexp>}, <xyzInnerBorders_nexp>}

Returns in <xyzTriangles_nexp> a two and a half dimensional xyz list of **triangle polygons** within a 2D

xy point cloud using the Delaunay triangulation. All triangle points are in clockwise order if you use screen coordinates. Using the right hand order the points are sorted in the counterclockwise direction.

If you use **World Coordinates** respectively the **Right-Hand Rule,** also known as a mathematically positive, the points are sorted in the counterclockwise direction.

The z components are added afterwards.

Returns also in <xyzMidpoints_nexp> a xyz list of **midpoints** of the triangle polygons.

The result can be used to refine (triple) the mesh by combining this result with the input points.

The list <xyzNormals_nexp> returns the **plain normal vectors** of the triangles.

The length of these vectors is specified by the optional <normal_length_nexp>. If it is 0, an empty list will be returned. Default is 1. Often the coordinates of plane normals are denoted as i, j, k instead of x, y, z.

The **input** list is defined by <xyzSource_nexp> containing the xyz values.

Make sure that you do not use points with equal xy coordinates, because this command does not check. But it is more robust, because if it checks a 0, 0, 0 normal it blocks this triangle and uses the first z value of this coordinate. But there is no guarantee that all possibilities are covered.

If your **outer border** of your mesh is not only convex, the optional xyz list <xyzOuterBorder_nexp> can be use to delete unwanted triangles outside the outer border.

Triangles of **inner contours** can be removed by the bundle <xyzInnerBorders_nexp> containing lists of xyz points which describe the inner polygons. Intersecting polygons give undesirable results.

Example:

```
List.create n, xyzPoints, xyzBorder
List.add xyzPoints,  0,0,0,  250,100,0,  500,0,0,  500,500,0,  250,400,0,
0,500,0
List.add xyzBorder, xyzPoints
List.add xyzPoints, 250,250,125
List.create n, triangles, midpoints, normals
Mesh.triangle.2.5d triangles, midpoints, normals, 1, xyzPoints , xyzBorder
```

40 Miscellaneous Commands

40.1 Headset

Syntax: Headset <state_nvar>, <type_svar>, <mic_nvar>

Reports if there is a headset plugged into your device, and returns data about the headset. The parameters are all names of variables that receive the data:

- <state_nvar>: 1.0 if a headset is plugged in, 0.0 if no headset is plugged in, and -1.0 if unknown.
- <type_svar>: A string describing the device type of the last headset known to your device.
- <mic_nvar>: 1.0 if the headset has a microphone, 0.0 if the headset does not have a microphone, and -1.0 if unknown.

If you plug in or unplug a headset, new information becomes available. Your program must run the **Headset** command again to get the update.

40.2 Pause

Syntax: Pause <ticks_nexp>

Stops the execution of the BASIC! program for <ticks_nexp> milliseconds. One millisecond = 1/1000 of a second. Pause 1000 will pause the program for one second. A pause can not be interrupted.

An infinite loop can be a very useful construct in your programs. For example, you may use it to wait for the user to tap a control on the screen. A tight spin loop keeps BASIC! very busy doing nothing. A **Pause**, even a short one, reduces the load on the CPU and the drain on the battery. Depending on your application, you may want to add a **Pause** to the loop to conserve battery power:

```
Do
 Pause 50
Until x <> 0
```

40.3 Swap

Syntax: Swap <nvar_a>|<svar_a>, <nvar_b>|<svar_b>

The values and in "a" and "b" numeric or string variables are swapped. The two variables must be of the same type.

40.4 Tone

Syntax: <frequency_nexp>, <duration_nexp> {, duration_chk_lexp}

Plays a tone of the specified frequency in hertz (cycles per second) for the specified duration in milliseconds.

The duration produced does not exactly match the specified duration. If you need to get an exact duration, experiment.

Each Android device has a minimum tone duration. By default, if you specify a duration less than this minimum, you get a run-time error message giving the minimum for your device. However, you can suppress the check by setting the optional duration check flag <duration_chk_lexp> to 0 (false). If you do this, the result you get depends on your device. You will not get a run-time error message, but you may or may not get the tone you expect.

40.5 Vibrate

Syntax: Vibrate <pattern_array[{<start>,<length>}]>,<nexp>

The vibrate command causes the device to vibrate in the specified pattern. The pattern is held in a numeric array (pattern_array[]) or array segment (pattern_array[start,length]).

The pattern is of the form: pause-time, on-time, …, pause-time, on-time. The values for pause-time and on-time are durations in milliseconds. The pattern may be of any length. There must be at least two values to get a single buzz because the first value is a pause.

If <nexp> = -1 then the pattern will play once and not repeat.
If <nexp> = 0 then the pattern will continue to play over and over again until the program ends.
If <nexp> > 0 then the pattern play will be cancelled.

See the sample program, **f21_sos.bas**, for an example of **Vibrate**.

41 NFC Commands

NFC (Near Field Communication) refers to contactless transmission of data between two devices or a so-called tag. With NFC, data can be transferred between an active device (e.g. Android smartphone or tablet with NFC) and a passive NFC tag. Small amounts of data (hundreds of bytes) can be sent over distances of a few centimeters. The NFC tag is powered by the active NFC device via electromagnetic induction during transmission. Radio waves at 13.56 MHz are used for data transmission.

Source: https://www.droidwiki.org/wiki/Near_Field_Communication

Please check if your device is able to read and write NFC Tags. Take care that a spare battery supports NFC. The manufacturer's original is the best choice.

Three tested SAMSUNG devices supported NFC well. A Nokia 5.1 with Android (One) 10 does not work, although it was specified.

If you do not have Bluetooth enabled (using the Android Settings Application) then the person running the program will be asked whether NFC should be enabled.

The following commands support only NFC cards and tags.

41.1 Nfc.read

Syntax: Nfc.read <bundle_nexp>

The bundle pointer <bundle_nexp> specifies a timer and returns the results of an NFC card or NFC tag.

The bundle keys and possible values are in the table below:

Key	Type	Value
_AskForNfc	numeric	Time in milliseconds to switch on if asked for NFC because it is not enabled. If 0, no question is asked. Default is 10000.
_Timer	numeric	Specifies a timer which returns, after given milliseconds, to the main program. Default is 0. In this case it will be returned only after reading a card or tag. Using a timer is strongly recommended.
_Error	String	Returns an error in the event of an error, otherwise an empty string.
_Id	String	Returns the ID of the card or tag. Note, this ID is not in all cases unique, because you can buy cards and tags with specified ID series.
_NfcData	String	Returns all NDEF text data records as a CSV string delimited by a LF (CHR$(10) or \n). If the string is UTF-8 encoded, it can be decoded by DECODE$ ("ASCII", ...$)
_Records	Array of String	Returns an Array of NDEF text data records. If the records are UTF-8 encoded, it can be decoded by DECODE$ ("ASCII", ...$)
_Tag	String	Returns the possible NFC types supported by the card or tag delimited by **", "** (comma and space).

Note that URIs, AppIDs and MIME records cannot be read because Android catches them first. For this issue the app NFC Tools (app id = com.wakdev.wdnfc) by wakdev.com is recommended.

Example:

```
Bundle.put nfcBun, "_Timer", 5000
Nfc.read nfcBun
Debug.on
Debug.dump.bundle nfcBun
Bundle.get nfcBun, "_Records", records$[]
Debug.dump.array records$[]
```

41.2 Nfc.write

Syntax: Nfc.write <bundle_nexp>

The bundle pointer <bundle_nexp> returns the results of an error and sets write settings of the NFC card or NFC tag.

The bundle keys and possible values are in the table below:

Key	Type	Value
_AskForNfc	numeric	Time in milliseconds to switch on if asked for NFC because it is not enabled. If 0, no question is asked. Default is 10000.
_Timer	numeric	Specifies a timer, which returns after given milliseconds to the main program. Default is 0. In this case it will be returned only after writing a card or tag. Using a timer is strongly recommended. In case of writing it should be equal or greater than 8000 milliseconds.
_Error	String	Returns an error in the event of an error, otherwise an empty string. If the writing fails no error is thrown. Please read the tag again to see that the writing is ok.
_NfcData	String	Submits one NDEF text data record as a string, if _Records is not specified. The string is UTF-8 encoded.
_Records	Array of String	Submits an Array of NDEF text data records to write. The records are UTF-8 encoded.
_Types	Array of String	Submits an Array of NDEF types. Default is **""** ("_Text"). Other types are "_App" and "_Uri". "_App" expects an AppId like "com.rfo.basicOli" "_Uri" expects an uri beginning with http(s)://…, file:///…, content://.. … If the array is shorter than the array in _Records the unspecified are from type **""** ("_Text").
_MessageLength	numeric	Returns the message length in Bytes.
_TagSpace	numeric	Returns the available NDEF memory space in Bytes.

Writing with Android devices, mostly smartphones, is not an easy task, because reading is fast, but when you write you need to be patient. Please put your card or tag on a desk and move your device down slowly.

If the writing fails, and reading does not return the desired result, you can use the app NFC Tools (app id = com.wakdev.wdnfc) by wakdev.com to fix it by writing a simple text record (after a formatting if needed) or copying an other tag.

Nfc.write is not able to overwrite _App or _Uri records. In this case, NFC Tools is also your helper when overwriting by a simple sentence.

Handle carefully your device at card or tag writing. Don't hurry. Success needs slow motion.

Otherwise, your tag can be destroyed forever.

Cards and tags with the NTAG 213 or NTAG 216 are recommended.

Example:

```
Array.load records$[], "First data record", "Second data record"
Bundle.put nfcBun, "_Timer", 8000
Nfc.write nfcBun
Bundle.get nfcBun, "_Error", error$
If error$ Then Print error$
```

42 Notify

42.1 Notify

Syntax: Notify <title_sexp>, <subtitle_sexp>, <alert_sexp>, <wait_lexp>{{{, <options_bundle_nexp>}, <notification_id_nvar>}, <notified_id_nvar>}

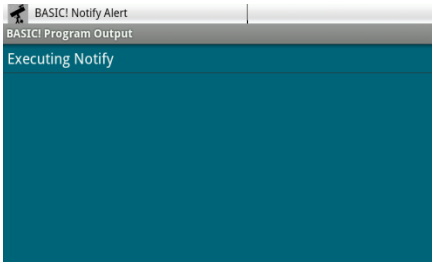
This command will cause a Notify object to be placed in the Notify (Status) bar. The Notify object displays the BASIC! app icon and the <alert_sexp> text. The user taps the Notify object to open the notification window. Your program's notification displays the <title_sexp> and <subtitle_sexp> text.

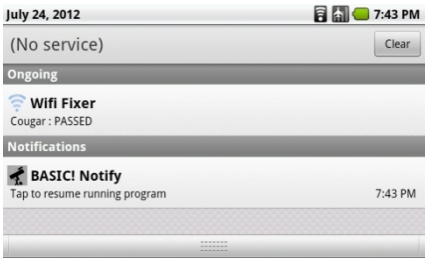
The code snippet and screenshots shown below demonstrate the placement of the parameter strings.

If <wait_lexp> is not zero (true), then the execution of the program will be suspended until the user taps the Notify object. In this case, the optional numeric variable <notified_id_nvar> returns the notification id of the touched notification. If <notified_id_nvar> returns -1, no notification touch was identified. If the value <wait_lexp> is zero (false), the program will continue executing.

The Notify object will be removed when the user taps or slides the object, or when the program exits. See also the bundle keys _AutoCancel and _Ongoing later.

```
Print "Executing Notify"
Notify "BASIC! Notify", "Tap to resume running program",~
"BASIC! Notify Alert", 1
! Execution is suspended and waiting for user to tap the Notify Object
Print "Notified"
```





The optional options bundle <options_bundle_nexp> controls the notification:

Key	Value	Description
_SmallIcon	Small icon file path	Sets the small icon, which will be used to represent the notification in the status bar. The platform template for the expanded view will draw this icon normally in the left. It is a special bitmap. The visible content is defined only by the alpha channel. http://romannurik.github.io/AndroidAssetStudio/icons-notification.html
_LargeIcon	Large icon file path	Add a large icon to the notification content view.

Key	Value	Description
		http://romannurik.github.io/AndroidAssetStudio/index.html
_Color	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	Min. Lollipop 5.0 (API 21)
_Progress	Max, progress, animated	Progress bar, Max → maximal (numeric value), progress → between 0 and maximal (numeric value), animated > 0 → true
_Sound	sound file path	Deprecated but Note: Beginning with Android 8.1 (API level 27), apps cannot make a notification sound more than once per second. If your app posts multiple notifications in one second, they all appear as expected, but only the first notification per second makes a sound. Beginning with Android 9 (API level 28) system sounds take over.
_AutoCancel	0 or > 0 (numeric)	Makes this notification automatically dismissed when the user touches or slides it. Default is true (> 0).
_Ongoing	0 or > 0 (numeric)	Sets whether this is an "ongoing" notification. Ongoing notifications cannot be dismissed by the user, so your application must take care of canceling them. They are typically used to indicate a background task that the user is actively engaged with (e.g., playing music) or is pending in some way and therefore occupying the device (e.g., a file download, sync operation, active network connection). Default is false (0).
_ShowWhen	0 or > 0 (numeric)	Beginning with Android 7 (API level 26) the notification time has to be set, if you want to display it. Default is false (0).

The optional numeric variable <notification_id_nvar> controls and returns a notification id.

<notification_id_nvar>	Control and Result
-2	A new notification id is created by the last nine digits of the current system time in milliseconds. Returns the created Id.
-1	Repeats and returns the last used notification id.
0	Default, returns 0
Given id	Uses the given id if possible. Otherwise 0 will be used and returned.

Note: the icon that appears in the Notify object will be the icon for the application in user-built APK.

Example:

```
! Bundle.put b, "_SmallIcon", "myIconAlphaChaneled.png"
Bundle.put b, "_LargeIcon", "cartman.png"
Bundle.put b, "_Sound", "whee.mp3"
maxVal = 1000
progress = 333
animate = 0
Bundle.put b, "_Progress", Int$(maxVal) + "," + Int$(progress) + "," + Int$
(animate)
Bundle.put b, "_Color", "255,0,0" % Red
myNnotify_ID_1 = -2
Notify "title1", "subtitle", "alert", 0 , b, myNnotify_ID_1
Print myNnotify_ID_1
Print

Bundle.put b, "_Sound", ""
Bundle.put b, "_Color", "0,255,0" % Green
myNnotify_ID_2 = -2
Notify "title2", "subtitle", "alert", 0 , b, myNnotify_ID_2
Print myNnotify_ID_2
Print

Bundle.put b, "_LargeIcon", "galaxy.png"
Bundle.put b, "_Color", "0,0,255" % Blue
myNnotify_ID_3 = -2
Notify "title3", "subtitle", "alert", 1 , b, myNnotify_ID_3, ret
Print myNnotify_ID_3
Print
Print ret
```

42.2 Notify.cancel

Syntax: Notify.cancel {<notification_id_nexp>}

Cancels all notifications triggered by your program by default.

Does not overwrite <wait_lexp> from the command Notify, because in this case the execution of the program is suspended until the user taps the Notify object.

If you use the optional <notification_id_nexp> with a number > 0 only a notification with this number will be canceled. If <notification_id_nexp> is 0 all notifications will be canceled like the default setting.

42.3 Notify.status

Syntax: Notify.status <lastStatus_nvar>

Returns last_Status of NOTIFY where:

0 = no wait (last NOTIFY was not waiting)
1 = system continued (e.g backkey) (notification not removed)
2 = tapped (user tapped notification)
3 = dismissed (user slides notification)
4 = cancelled (by NOTIFY.Cancel)

43 Permissions

Permissions are special privileges that apps must ask for if they want to access sensitive media on your Android device.

Android devices contain much personal information, like the exact location, contact data, cameras that can record the user, etc.. Apps can not just use these unless the user gives permission. Beginning with Android 6.0 "Marshmallow" API level = 23, permission handling is more defensive.

But if your program is running on a lower API level, there are no restrictions.

Permissions which have dangerous protection level **have be requested first** in Basic's or your application's manifest. By default these are not be granted the first time your app runs after the first installation.

Note that normal permissions are granted at install time if requested in the manifest.

An app's permissions can be checked at any time. If the app is already installed, go to Settings → Apps and locate the app you want to examine. Tap an app, and on the About screen click the Permissions box or scroll down to find the list. Here you can see everything the app asks for. For a programmed link see **App.settings**.

File access to Resources, Assets and the Internal file directory should not need any permissions because these are protected areas.

It is a good practice to check the needed permissions at the first application start.

To prevent irritating the user, explain your requested permissions in detail, because if you trigger the **Device** command with a bundle, you need Phone permissions. But if you do not want to make a call, you should describe that you only need device information.

Do not be surprised if the user later changes one or more of the required permissions.

Maybe the user denied the permissions for camera and microphone for privacy reasons.

So you should take appropriate precautions, such as using **Permission.get**.

If you want to compile your program as an APK, **Permission.checkPath** is useful. You can check if the given file path is a candidate for a file permission request.

Note that permissions in conjunction with Internal Directories on an External Directory or on removable SD-cards were not tested until now.

When delivering your program as an APK, ask as soon as possible for all needed permissions. Use assets and the internal directory for easy file permission control, because in this case you do not have to request for permissions.

Permissions used by AndroidManifest.xml	Dangerous Level
ACCESS_COARSE_LOCATION	✓
ACCESS_FINE_LOCATION	✓
ACCESS_LOCATION_EXTRA_COMMANDS	
ACCESS_MOCK_LOCATION	
ACCESS_NETWORK_STATE	
ACCESS_SUPERUSER	

Permissions used by AndroidManifest.xml	Dangerous Level
ACCESS_WIFI_STATE	
ACTIVITY_RECOGNITION	✓
BLUETOOTH_ADMIN	
BLUETOOTH	
BODY_SENSORS	✓
CALL_PHONE	✓
CAMERA	✓
INSTALL_SHORTCUT	
INTERNET	
KILL_BACKGROUND_PROCESSES	
READ_CONTACTS	✓
READ_EXTERNAL_STORAGE	✓
READ_PHONE_STATE (Android 10-) or READ_PHONE_NUMBERS (Android 11+)	✓
READ_SMS	✓
READ_USER_DICTIONARY	
RECEIVE_SMS	✓
RECORD_AUDIO	✓
RUN_SCRIPT (Only to detect over APP.settings!)	✓
SEND_SMS	✓
UNINSTALL_SHORTCUT	
VIBRATE	
WAKE_LOCK	
WRITE_EXTERNAL_STORAGE	✓
WRITE_SETTINGS	

Note that WRITE_EXTERNAL_STORAGE includes the READ_EXTERNAL_STORAGE permission.

43.1 Permission.automatic

Syntax: Permission.automatic <auto_nvar>

OliBasic has automatic permission detection and request. This can be switched to OFF by setting <auto_nvar> to 0, or to ON by setting <auto_nvar> to 1 (default).

If <auto_nvar> is set to 2, it is switched to ON but with no file permissions.

It is only an option to switch to OFF in case of a massive data file transfer because of system delay. If possible, **Permission.ignore** is a better option.

43.2 Permission.checkPath

Syntax: Permission.checkPath <nvar>, <checkPath_sexp>, <dump_nexp>

If <nvar> is set to 1, the file name <checkPath_sexp> is checked for STORAGE permission as it relates to the shared external storage (READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE).

If <dump_nexp> > 0, detailed information is printed.

43.3 Permission.get

Syntax: Permission.get <granted_lvar>, <permission_sexp>

Returns <granted_lvar> = 1 if the permission given by <permission_sexp> is granted. Otherwise, <granted_lvar> returns 0.

43.4 Permission.ignore

Syntax: Permission.ignore <file_path_Array$[]>

Dealing with file access permissions requires some care. Assets and Internal directories should be ignored for permission checking. File names including "files:///data/", "Android/data/(Application's package name)" and "asset://" will be ignored by default.

If you deal with relative file paths, it could be useful to add your own exceptions.

43.5 Permission.request

Syntax: Permission.request <permission_sexp> | Array$[]

Requests permissions to be granted to the running Basic-engine or your application(APK). For Android 6.0 Marshmallow, API level = 23 and later.

Level Dangerous Permission Group	Related Permissions
CALENDAR Used for runtime permissions related to user's calendar.	READ_CALENDAR WRITE_CALENDAR
CALL_LOG (Since Android 9) This permission group gives control and visibility to apps that need access to sensitive information about phone calls, such as reading phone call records and identifying phone numbers.	PROCESS_OUTGOING_CALLS READ_CALL_LOG WRITE_CALL_LOG
CAMERA Used for permissions that are associated with accessing camera or capturing images/video from the device.	CAMERA
CONTACTS Used for runtime permissions related to contacts and profiles on this device.	READ_CONTACTS WRITE_CONTACTS GET_ACCOUNTS
LOCATION Used for permissions that allow accessing the device location. OliBasic supports by default only ACCESS_FINE_LOCATION (GPS and Network).	(ACCESS_COARSE_LOCATION) ACCESS_FINE_LOCATION

Level Dangerous Permission Group	Related Permissions
MICROPHONE Used for permissions that are associated with accessing microphone audio from the device.	RECORD_AUDIO
PHONE Used for permissions that are associated telephony features.	ANSWER_PHONE_CALLS READ_PHONE_STATE READ_PHONE_NUMBERS CALL_PHONE ADD_VOICEMAIL USE_SIP (Until Android 8.1 PROCESS_OUTGOING_CALLS READ_CALL_LOG WRITE_CALL_LOG)
SENSORS Used for permissions that are associated with accessing body or environmental sensors.	BODY_SENSORS
SMS Used for runtime permissions related to user's SMS messages. SMS will be only supported until Android version 10, because of the Android security management.	READ_SMS SEND_SMS RECEIVE_SMS RECEIVE_WAP_PUSH RECEIVE_MMS
STORAGE Used for runtime permissions related to the shared external storage.	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE
Random Terminal Scripts	Only to control over APP.settings

Note that not all table permissions are used in conjunction with Basic.

Normal permissions are granted at install time if requested in the manifest.

If your program does not have the requested permissions, the user will be presented with UI for accepting them.

Note that requesting a permission does not guarantee it will be granted and your program should be able to run without having this permission.

This command may start an activity allowing the user to choose which permissions to grant and which to reject. Hence, you should be prepared that your activity may be paused and resumed. **Further, granting some permissions may require a restart of you application!**

When checking whether you have a permission you should use **Permission.get**.

Google Reference:

Calling this command for permissions already granted to your app would show UI to the user to decide whether the app can still hold these permissions. This can be useful if the way your app uses data guarded by the permissions changes significantly.

But after experiences:

The UI is only created, if the permission is not granted.

Note that if you use the single string only, if will request one permission.

A second one or more without a pause will be ignored. Use instead an array.

Example:

```
Permission.request “READ_PHONE_STATE”
```

44 Phone Commands

44.1 MyPhoneNumber

Syntax: MyPhoneNumber <svar>

The phone number of the Android device will be returned in the string variable. If the device is not connected to a cellular network, the returned value will be uncertain.

44.2 Phone.call

Syntax: Phone.call <sexp>

The phone number contained in the string expression will be called. Your device must be connected to a cellular network to make phone calls.

44.3 Phone.dial

Syntax: Phone.dial <sexp>

Open the phone dialer app. The phone number contained in the string expression will be displayed in the dialer. Alphabetic characters in the string will be converted to digits, as if the corresponding key of a phone pad had been touched.

44.4 Phone.rcv.init

Syntax: Phone.rcv.init

Prepare to detect the state of your phone. If you want to detect phone calls using **Phone.rcv.next**, or you want **Phone.info** to attempt to report signal strength, you must first run this command.

Phone.rcv.init starts a background "listener" task that detects changes in the phone state. There is no command to disable this listener, but it is stopped when your program exits.

44.5 Phone.rcv.next

Syntax: Phone.rcv.next <state_nvar>, <number_svar>

The state of the phone will be returned in the state numeric value. A phone number may be returned in the string variable.

State = 0. The phone is idle. The phone number will be an empty string.

State = 1. The phone is ringing. The phone number will be in the string.

State = 2. The phone is off hook. If there is no phone number (an empty string) then an outgoing call is being made. If there is a phone number then an incoming phone call is in progress.

States 1 and 2 will be continuously reported as long the phone is ringing or the phone remains off hook.

45 Program Control, Execution and Status Commands

45.1 Include

Syntax: Include FilePath

Before the program is run, the BASIC! preprocessor replaces any **Include** statements with the text from the named file. You can use this to insert another BASIC! program file into your program at this point. The program is not yet running, therefore the File Path cannot be a string expression.

You may include a file only once. If multiple **Include** statements name the same file, the preprocessor inserts the contents of the file in place of the first such **Include** statement and deletes the others. This prevents Out-Of-Memory crashes caused by an **Include** file including itself.

```
Include functions/DrawGraph.bas
```

inserts the code from the file "<pref base drive>/rfo-basic/source/functions/DrawGraph.bas" into the program.

The File Path may be written without quotation marks, as in the example above, or with quotes:

```
Include "functions/DrawGraph.bas"
```

If present, the quotes prevent the preprocessor from forcing the File Path to lower-case. Normally, this does not change how BASIC! behaves, because the file system on the SD card is case-insensitive. **DrawGraph.bas** and **drawgraph.bas** both refer to the same file.

However, if you build your program into a standalone Android application, you can use virtual files in the Android **assets** file system. File names in **assets** are case-sensitive, so you may need to use quotes with the **Include** File Path.

Because **Include** is processed before your program starts running, it is not affected by program logic:

```
If 0
  Include functions/DrawGraph.bas
EndIf
```

In the above example, the contents of the included file will replace the **Include** statement in the body of the **If**/**EndIf**, but the statements are never executed because **If 0** is always false.

However, an **Include** in a single-line **If** is ignored:

```
If x Then Include functions/DrawGraph.bas Else Print "bad!"
```

In this example, the file is not inserted in place of the **Include** statement.

45.2 Program.animations

Syntax: Program.animations <bundle_nexp>

This command uses the bundle at <bundle_nexp> to control the swipe animations in modern Android systems. Currently, the bundle keys are: _GrOnStart, _GrOnStop, _SelectOnStart, and _TextInputOnStart. If the parameter equals 0.0, swipe animation is prevented. If the parameter is not 0.0, swipe animation will be work if supported by the system. Default is 1.0.

Example

```
!  Prevents the swipe animation at calling the Graphic mode
```

```
Bundle.put pa, "_GrOnStart", 0
!  Prevents the swipe animation at returning the Console
Bundle.put pa, "_GrOnStop", 0
!  Prevents the swipe animation at calling the TextInput mode
Bundle.put pa, "_TextInputOnStart", 0
Program.animations pa

Bundle.put ctb, "_TitleShow", 0
Console.title "", ctb
Bundle.put clb, "_BackgroundColor", "_100,Orange"
Console.layout clb
Gr.open "_100,Orange", 1,1
```

45.3 Program.info

Syntax: Program.info <nexp>|<nvar>

Returns a Bundle that reports information about the currently running program. If you provide a variable that is not a valid Bundle pointer, the command creates a new Bundle and returns the Bundle pointer in your variable. Otherwise it writes into the Bundle your variable or expression points to.

The bundle keys and possible values are in the table below:

Key	Type	Value
_BasPath	String	Full path + name of the program currently being executed. The path is relative to BASIC!'s "source/" directory. See also **File.root**.
_BasName	String	Name of the program currently being executed.
_SysPath	String	Full path to the BASIC!'s private file storage directory. The path is relative to BASIC!'s "data/" directory.
_UserApk	Numer (Logical)	Returns 1.0 (true) if the current program is being run from a standalone user-built APK. Returns 0.0 (false) if the program is being from from the BASIC! Editor or a Launcher Shortcut.
_LauncherStart	Numeric (Logical)	Returns 1.0 (true) if the current program is being run from a Launcher Shortcut or an intent. Else returns 0.0 (false).
_PackageName	String	package_id
_AppVersion	String	Application Version (from the AndroidManifest.xml).
_AppVersionCode	Numeric	Application Version Code (from the AndroidManifest.xml).
_Build	String	Returns the Build of the underlying OliBasic Version.
_MyProcessId	String	my_process_id
_TotalUserRam	Numeric	The total memory accessible by the kernel. This is basically the RAM size of the device, not including below-kernel fixed allocations like DMA buffers, RAM for the baseband CPU, etc.
_AvailableRam	Numeric	The available memory on the system. This number should not be considered absolute: due to the nature of the kernel, a significant portion of this memory is actually in use and needed for the overall system to run well.
_ThresholdRam	Numeric	The threshold of available memory at which we consider memory to be low and start killing background services and other non-extraneous processes.
_HeapLimit	Numeric	Return the approximate per-application memory class of the current device. This gives you an idea of how hard a memory limit you should impose on your application to let the overall system work best.
_NativeHeap	Numeric	Returns the (dynamic) size of the native heap.

Key	Type	Value
_NativeHeapAllocated	Numeric	Returns the amount of allocated memory in the native heap.
_NativeHeapFree	Numeric	Returns the amount of free memory in the native heap.

For example, assume:

- You are using the default <pref base drive>
- You downloaded a file called "my_program.bas" to the standard Android Download directory.
- You used the BASIC! Editor to load and run the downloaded program.

Then the returned values would be as follows:

Key	Value
_BasPath	../../Download/my_program.bas
_SysPath	../../../../../data/data/com.rfo.basic
_BasName	my_program.bas
_UserApk	0.0
_PackageName	com.rfo.basic
_MyProcessId	21615
_AppVersion	3.00
_AppVersionCode	3030.0
_Build	3.00 Preview30n
_TotalUserRam	2759.0 in megabytes.
_AvailableRam	1370.0 in megabytes.
_ThresholdRam	216.0 in megabytes.
_HeapLimit	192.0 in megabytes.
_NativeHeap	15.89... in megabytes.
_NativeHeapAllocated	12.19... in megabytes.
_NativeHeapFree	3.69... in megabytes.

SysPath is of particular interest to you if you build a BASIC! program as an application in a standalone APK. BASIC! normally keeps programs and data in its *base directory* (see **Working with Files**). The base directory is in public storage space (external). BASIC! programs also have access to a private (internal) storage area. Your program can create a subdirectory within the SysPath directory and store private files there. Note that if you uninstall BASIC!, any files in private storage will be deleted.

You can extent the heap size by inserting in the APK's AndroidManifest.xml file the following line:

```
android:largeHeap="true";
```

45.4 Run

Syntax: Run <filename_sexp>{, <data_sexp>}

This command will terminate the running of the current program and then load and run the BASIC! program named in the filename string expression. The filename is relative to BASIC!'s "source/" directory. If the filename is "program.bas" and your <pref base drive> is "/sdcard" (the default), then the file "/sdcard/rfo-basic/source/program.bas" will be executed.

If the filename parameter is omitted, and the currently executing program has a name, then the program restarts. The program does not have a name if you run from the BASIC! Editor without first saving the program; in this case **Run** terminates your program with a syntax error.

The optional data string expression provides for the passing of data to the next program. The passed data can be accessed in the next program by referencing the special variable, **##$**.

Run programs can be chained. A program loaded and run by means of the **Run** command can also run another program file. This chain can be a long as needed.

When the last program in a **Run** chain ends, tapping the BACK key will display the original program in the BASIC! Editor.

When a program ends with an error, the Editor tries to highlight the line where the error occurred. If the program with the error was started by a **Run** command, the Editor does not have that program loaded. Any highlighting that may be displayed is meaningless.

If the first parameter is "" or not given, a program with the current file path will load and run (if not compiled maybe with different code). In APK mode, the App will be relaunched.

Note: "file://" + <full_path_svar> can now be used for absolute file paths.

If <filename_sexp> is "" or is not given, the current program itself will restart. Be careful for not to get caught in an endless loop.

Example:

```
!   Run "Files.bas"
!   Run "Files.bas", ""
!   Run %Endless loop?
!   Run "" %Endless loop?
!   Run " " %File Not found
Program.info bdi
Bundle.get bdi, "BasName", BasName$
Print BasName$
File.root aPath$,"_Source"
If ##$ <> "1" Then
 ! Run , "1"

 ! RUN "file://" + aPath$ + "/" + BasName$, "1"
EndIf
! Run "../source/Files.bas", ""
File.root dp$, "_Source"
Print dp$

Run  "file://"+dp$ + "/" + "Files.bas"
```

45.5 Shell

Syntax: Shell <result_svar>, <command_sexp>

Opens a **SHELL** to execute system commands <command_sexp>. Unlike **System.open**, the working directory is set to "**root**". The command waits for a result and places it in <result_svar>.

Note, often a shell script (*.sh) is needed for execution. You can build these scripts with BASIC! commands and functions, and store them as files in the internal file system.

Example:

```
File.root path$
d$ = "cat " + path$ + "/" + "htmldemo1.html"
Shell r$, d$
Print r$
```

45.6 Version$

Syntax: Version$()

Returns the version number of BASIC! as a string.

46 Program Flow Statements

46.1 Do / Until

Syntax: Do / Until <lexp>

```
Do
  <statement>
  …
  <statement>
Until <lexp>
```

The statements between **Do** and **Until** will be executed until <lexp> is true. The <statement>s will always be executed at least once.

Do-Until loops may be nested to any level. Any encountered **Until** statement will apply to the last executed **DO** statement.

It is BAD practice to have a **GoTo** that exits the **Do** / **Until** loop. This can create subtle bugs which BASIC! can help you find. If debug is on, and your program is still in a **Do** loop when it ends, BASIC! shows a run-time error: "Program ended with DO without UNTIL".

46.1.1 D_U.continue

Syntax: D_U.continue

If this statement is executed within a **Do-Until** loop, the rest of the current pass of the loop is skipped. The **Until** statement executes immediately.

46.1.2 D_U.break

Syntax: D_U.break

If this statement is executed within a **Do-Until** loop, the rest of the current pass of the loop is skipped and the loop is terminated. The statement immediately following the **Until** will be executed.

46.2 For - To - Step / Next

Syntax: For - To - Step / Next

```
For <nvar> = <nexp_1> To <nexp_2> {Step <nexp_3>}
  <statement>
  ...
  <statement>
Next {<nvar>}
```

Initially, <nvar> is assigned the value of <nexp_1> and compared to <nexp_2>. {Step <nexp_3>} is optional and may be omitted. If omitted then the **Step** value is 1.

If <nexp_3> is positive then
    if <nvar> <= <nexp_2> then
        the statements between the **For** and **Next** are executed.

If <nexp_3> is negative then
    if <nvar> >= <nexp_2> then
        the statements between the **For** and **Next** are executed.

When the **Next** statement is executed, <nvar> is incremented or decremented by the **Step** value and the test is repeated. The <statement>s will be executed as long as the test is true. Each time, <nvar>

is compared to the original value of <nexp_2>; <nexp_2> is not re-evaluated with each **Next**.

Because the keywords **To** and **Step** are in the middle of the line with expressions that may include variables, it is possible to confuse BASIC!. Remember that the interpreter does not see any spaces you put between variables and keywords. **For a To m** is seen as **foratom**. If there is any possibility of confusion, use parentheses to tell BASIC! that a name is a variable:

```
For  WinTop  To WinBot    % ERROR: interpreted as "FOR win TO ptowinbot"
For (WinTop) To WinBot    % interpreted as intended
```

For-Next loops can be nested to any level. When **For-Next** loops are nested, any executed **Next** statement will apply to the currently executing **For** statement. This is true no matter what the <nvar> coded with the **Next** is. For all practical purposes, the <nvar> coded with the **Next** should be considered to be nothing more than a comment.

It is BAD practice to have a **GoTo** that exits the **For** / **Next** loop. This can create subtle bugs which BASIC! can help you find. If debug is on, and your program is still in a **For** loop when it ends, BASIC! shows a run-time error: "Program ended with FOR without NEXT".

46.2.1 F_N.continue

Syntax: F_N.continue

If this statement is executed within a **For-Next** loop, the rest of the current pass of the loop is skipped. The **Next** statement executes immediately.

46.2.2 F_N.break

Syntax: F_N.break

If this statement is executed within a **For-Next** loop, the rest of the current pass of the loop is skipped and the loop is terminated. The statement immediately following the **Next** will be executed.

46.3 For / Next

Syntax: For <nvar>, Array[]|Array$[] / Next

Initiates a **For/Next** loop. The index given by <nvar> starts with 1 and counts up until the total length of an array given by Array[] or Array$[]. If the array has more than one dimension, the index counts up column by column.

It is BAD practice to have a **GoTo** that exits the **For** / **Next** loop. This can create subtle bugs which BASIC! can help you find. If debug is on, and your program is still in a **For** loop when it ends, BASIC! shows a run-time error: "Program ended with FOR without NEXT".

46.4 If / Then / Else / ElseIf / EndIf

Syntax: If / Then / Else / ElseIf / EndIf

The **If** commands provide for the conditional execution of blocks of statements. (Note: the braces { } are not part of the command syntax. They are used only to show parts that are optional.)

```
If <condition> { Then }
    <statement>
    <statement>
...
    <statement>
{ ElseIf<condition> { Then }
    <statement>
    <statement>
```

```
...
    <statement> }
{ Else
    <statement>
    <statement>
...
    <statement> }
EndIf
```

If commands may be nested to any depth. That is, any <statement> in a block may be a full **If** command with all of its own <statement> blocks.

See the Sample Program file **F04_if_else.bas** for working examples of the **If** command.

It is BAD practice to have a **GoTo** that exits the **If** / **EndIf** loop.

46.5 If ... Then ... Else

Syntax: If ... Then ... Else

If your conditional block(s) contain(s) only one statement, you may use a simpler form of the **If** command, all one line:

```
If <condition> Then <statement> { Else <statement> }
```

In this form, **Then** is required, and there is no **ElseIf** or **EndIf**.

This form does not nest: neither <statement> may be an **If** command.

Because the single statements are not treated as blocks, this is the preferred form if either of the embedded statements is a **Break**, **Continue**, or **GoTo**.

You may replace either <statement> with multiple statements separated by colon (":") characters. If you do this, the set of multiple statements is treated as a block, and the single-line **If...Then...Else** becomes an **If/Then/Else/EndIf**. These two lines are exactly equivalent:

```
If (x > y) Then x = y : Print a$ Else y = x : Print b$
If (x > y) : x = y : Print a$ : Else : y = x : Print b$ : EndIf
```

Please note, if you wish to use colon-separated statements in this form of **If/Then/Else**, then you must be careful to put spaces around the keywords **Then** and **Else**. Spaces are not significant to the BASIC! interpreter, but they are needed by the preprocessor that converts the single-line **If** with multi-statement blocks into a multi-line **If** with an **EndIf**.

46.6 Switch Commands

The Switch commands may be used to replace nested if-then-else operations.

```
Sw.begin a
Sw.case 1
  <statement1>
  …
  <statement2>
  Sw.break

Sw.case 2, 4
  <statement3>
  …
  <statement4>
  Sw.break

Sw.case < 0
```

```
    <statement5>
    …
    <statement6>
    Sw.break

  Sw.default
    <statement7>

  Sw.end
```

The value of the argument of **Sw.begin** is compared to the argument of each **Sw.case** in order. If any **Sw.case** matches the **Sw.begin**, the statements following the matching **Sw.case** is executed; if no **Sw.case** matches, the statements after **Sw.default** are executed. Once BASIC! starts to execute the statements of a **Sw.case** or **Sw.default**, it continues execution until it finds a **Sw.break** or the **Sw.end**, ignoring any other **Sw.case** or **Sw.default** it may encounter. **Sw.break** causes a jump to the **Sw.end**.

In the example:

- if the value of **a** is 1, then <statement1> through <statement2> execute
- if the value of **a** is 2 or 4, then <statement3> through <statement4> execute
- if the value of **a** is less than 0, then <statement5> through <statement6> execute
- if **a** has any other value (3, or more than 4) then <statement7> executes.

A **Sw.begin** must be followed by a **Sw.end**, and all of the **Sw.case** and the **Sw.default** (if there is one) for the same switch must appear between them. A set of switch commands is treated as a single unit, just as if BASIC! were compiled.

It is BAD practice to have a **GoTo** that exits a Switch structure.

46.6.1 Nesting Switch Operations

Switches can be nested. The block of statements following a **Sw.case** or **Sw.default** may include a full set of switch commands. The nested switch begins with another **Sw.begin** and ends with another **Sw.end**, with its own **Sw.case** and **Sw.default** statements in between. You can nest other switches inside nested switches, for as many levels as you want.

If you prefer, you may put the inner switch operations in a labeled **GoSub** routine or a User-defined Function, and put the **GoSub** or function call in the **Sw.case** or **Sw.default** block. This may make your code easier to read, and it will also make the initial scan of the switch a little faster.

46.6.2 Sw.begin

Syntax: Sw.begin <exp>

Begins a switch operation. BASIC! scans forward until it reaches a **Sw.end**, locating all **Sw.case**, **Sw.break**, and **Sw.default** statements between the **Sw.begin** and the **Sw.end**.

The numeric or string expression <exp> is evaluated. Its value is then compared to the expression(s) in each **Sw.case** statement, in order. BASIC! uses to result of the compares to decide which statement to execute next.

IF this condition is met:	THEN jump to the statement
One or more Sw.Case statement(s) match the Sw.Begin	after the *first* matching **Sw.case**.
No Sw.Case matches AND a Sw.default exists	after the **Sw.default**.
No Sw.Case matches AND no Sw.default exists	after the **Sw.end**.

There are two forms of **Sw.case**. You may freely mix both forms. Each form defines what it means to "match" **Sw.begin**. Only the first matching **Sw.case** has any effect.

46.6.3 Sw.case

Syntax: Sw.case <exp>, ...

or

Syntax: Sw.case <op><exp>

The first form of **Sw.case** provides a list of one or more expressions. A **Sw.case** of this form matches the **Sw.begin** if the value of at least one of the expressions *exactly equals* the value of the **Sw.begin** parameter. The type of the **Sw.begin** and **Sw.case** parameter(s), numeric or string, must match.

The second form can take only one expression <exp>, but it lets you specify a different logical operator <op>. You may use any of these comparison operators:

```
<   <=   >   >=   <>
```

For example:

```
Sw.begin a
Sw.case < b          % This Sw.case matches if a < b
```

The expression <exp> may be arbitrarily complex. The whole expression is evaluated as if written:

```
<value of Sw.begin argument> <op> <exp>
```

Operator precedence is applied as usual.

46.6.4 Sw.break

Syntax: Sw.break

This statement may be used to terminate the block of statements that follows **Sw.case** or **Sw.default**. The **Sw.break** causes BASIC! to jump forward to the **Sw.end** statement, skipping everything between.

If no **Sw.break** is present in a particular **Sw.case** then subsequent **Sw.case**s will be executed until a **Sw.break** or **Sw.end** is encountered.

46.6.5 Sw.default

Syntax: Sw.default

This statements acts like a **Sw.case** that matches any value. If any **Sw.case** matches the **Sw.begin** value, then the **Sw.default** is ignored, even if the matching **Sw.case** is after the **Sw.default**.

A switch is not required to have a **Sw.default**, but it must not have more than one. A second **Sw.default** in the same switch is a syntax error.

46.6.6 Sw.end

Syntax: Sw.end

The **Sw.end** terminates a switch operation. **Sw.end** must eventually follow a **Sw.begin**.

46.7 While / Repeat

Syntax: While <lexp> / Repeat

```
While <lexp>
  <statement>
  …
  <statement>
Repeat
```

The <statement>s between the **While** and **Repeat** will be executed as long as <lexp> evaluates as true. The <statements>s will not be executed at all if <lexp> starts off false.

While-Repeat loops may be nested to any level. When **While-Repeat** are nested, any executed **Repeat** statement will apply to inner most **While** loop.

It is BAD practice to have a **GoTo** that exits a **While** / **Repeat** loop.

46.7.1 W_R.continue

Syntax: W_R.continue

If this statement is executed within a **While-Repeat** loop, the rest of the current pass of the loop is skipped. The **Repeat** statement executes immediately.

You can exit a **While** loop without **Repeat** or **W_R.break**. As with **For-Next** loops, this can create subtle bugs, and BASIC! can help you find them. If debug is on, and your program is still in a **While** loop when it ends, BASIC! shows a run-time error: "Program ended with WHILE without REPEAT".

46.7.2 W_R.break

Syntax: W_R.break

If this statement is executed within a **While-Repeat** loop, the rest of the current pass of the loop is skipped and the loop is terminated. The statement immediately following the **Repeat** will be executed.

46.8 Labels, GoTo, GoSub, and Return

A **GoTo** statement is a one-way jump to another place in your program, identified by a **Label**. The program goes to the **Label** and continues execution there.

A **GoSub** is similar, except that the **Label** is the beginning of a "Subroutine". The program goes to the **Label**, and executes there until it reaches a **Return** statement. Then it "returns", going back to where it came from: the line after the **GoSub** statement.

Extensive use of the **GoTo** command in your program should be generally avoided. It can make code hard to read and harder to debug. Instead, you should use structured elements like **Do…Until**, **While…Repeat**, etc. in conjunction with the **Break** and **Continue** statements.

It is especially serious to use **GoTo** commands inside an **If…Else…EndIf**, **For…Next**, or other structured block, jumping to code outside of the block. Doing this consumes system resources and may corrupt BASIC!'s internal data structures. This practice may lead your program to a run-time error:

```
Stack overflow. See manual about use of GOTO.
```

46.8.1 Label

A label is a word followed by the colon ":" character. Label names follow the same conventions as variable names, except that a label must not start with a BASIC! command keyword.

You may put a label on a line with other commands. Use two colons: one to signify that the word is a label, and a second to separate the label from the other command(s) on the line.

```
Here: : If ++a < 5 Then GoTo Here Else Print a
```

This program prints `5.0` and then stops.

The colon signifies that the word is a label, but it is not part of the label. Use the colon where the label is defined. Do not use it in the **GoTo** or **GoSub** that jumps to the label.

For example:

```
This_is_a_Label:

@Label#3:

Loop:          % The command "GoTo Loop" jumps to this line
  <statement>
  …
  <statement>
GoTo Loop
```

46.8.2 GoTo

Syntax: GoTo <label>

or

Syntax: GoTo <index_nexp>, <label>...

For the first form of the **GoTo** statement, the next statement to be executed is the statement following <label>.

The second form is called a "computed **GoTo**". The index expression is evaluated, rounded to the nearest integer, and used as an index into the list of labels. The program jumps to the statement after the indexed label. If the index does not select any label, the program continues at the statement after the **GoTo**.

It is BAD practice to include a GoTo statement inside a structured block (If / EndIf,

Example:

```
d = Floor(6 * Rnd() + 1)        % roll a six-sided die
GoTo d, Side1, Side2, Side3, Side4, Side5, Side6
Print "Welcome back!"
End

Side1:
  <statements>
.
.
.
Side6:
  <statements>
```

46.8.3 GoSub / Return

Syntax: GoSub <label> / Return

or

Syntax: GoSub <index_nexp>, <label>...

For the first form of the **GoSub** statement, the next statement to be executed is the statement following <label>.

The statements following the line beginning with <label> will continue to be executed until a **Return** statement is encountered. Execution will then continue at the statement following the **GoSub** statement.

Example:

```
Message$ = "Have a good day"
```

```
GoSub xPrint
Print "Thank you"
<statement>
…
<statement>
End

xPrint:
  Print Message$
Return
```

This will print:

```
Have a good day
Thank you
```

The second form is called a "computed GoSub". The index expression is evaluated, rounded to the nearest integer, and used as an index into the list of labels. The program jumps to the statement after the indexed label. When the next **Return** instruction executes, the program returns to the statement after this **GoSub**.

If the index does not select any label, the program continues to the statement after the **GoSub**. No subroutine is executed, and no **Return** statement is expected.

Example:

```
d = Floor(6 * Rnd() + 1)        % roll a six-sided die
GoSub d, Side1, Side2, Side3, Side4, Side5, Side6
Print "Welcome back!"
End

Side1:
  <subroutine for side 1>
Return
.
.
.
Side6:
  <subroutine for side 6>
Return
```

46.9 End

Syntax: End {<msg_sexp>}

Prints a message and stops the execution of the program. The default message is "END". You can use the optional <msg_sexp> argument to specify a different message. The empty string ("") prints nothing, not even a blank line. The **End** statement always stops execution, even if the statement has an error.

End statements may be placed anywhere in the program.

46.10 Exit

Syntax: Exit

Causes BASIC! to stop running and exit to the Android home screen.

47 Provider Commands

Android supports document and file **providers**. A provider is a tool that can used independently of a running app. Providers can be used to share files across the borders of Android's Scoped Storage. Providers manage documents and expose them to the Android system for sharing. Providers run in an independent instance and can be called by an internal or external request. An independent instance will execute regardless if your app is running or not. Also if you stop the running at the System App Info Preferences. And that is also the reason that you can control the providers only indirectly over Shared Preferences from your app.

Unfortunately, this opens a large back door for unwanted data and hidden manipulation, as this instance also has access to all public static methods of the internal Java code. Particularly dangerous if "true" is entered under AndroidManifest.xml <Provider android: exported = "true".

But that is the case if you follow Google's advice and you provide documents under Scoped Storage by a Documents (Adoc) Provider or any other Provider for public use.

To prevent this issue, the command **Provider** offers appropriate switches / settings.

As already mentioned, and according to Google's advice, you should use providers starting with scoped storage at the latest.

Are there alternatives? Get Rid of Providers:

- If you have data with roughly less than one Mb in size, you can use the data transfer over intents. Maybe little zipping will help.
- A simple nice internal file picker is more secure than a provider for internal use.
- A simple Ftp(s) server could also do the task if you are in a network or your device creates a WLAN hotspot.

If you want to use a Provider, an Adoc Provider is the best choice for security. But this is not a solution if you want to provide a file, maybe for a PDF viewer or a mail client. In this case the File Provider is a better choice.

47.1 Provider

Syntax: Provider <bundle_pointer_nexp>

Sets permissions and values. These settings are stored as default values and used at start as an independent provider called by a the same or different app.

If you provide a variable that is not a valid Bundle pointer, the command creates a new Bundle and returns the Bundle pointer in your variable. Otherwise, it writes into the Bundle that your variable or expression points to.

The bundle keys and possible values are in the table below:

Key	Type	Value
_FileP_Read	numeric	Sets the read permission if > 0 within the File Provider. Default is 0.
_FileP_Write	numeric	Sets the write permission if > 0 within the File Provider. Default is 0. If other apps should be give write permissions these apps should use a **full** exception handling.

Key	Type	Value
_FileP_Dir	numeric	Sets the get-directory permission if > 0 within the File Provider. Default is 0. FileProviderDir.txt returns the updated directory content. FileProviderDateDir.txt returns the updated directory content with a timestamp at the begin of each record. If it is 0 both files return an empty string. Note, that _FileP_Read have to be > 0 also.
_FileP_DirOut	numeric	If > 0, directories will be returned also. Default = 0
_ADocP_Read	numeric	Sets the read permission if > 0 within the ADoc Provider. Default is 0.
_ADocP_Write	numeric	Needs additionally also permissions from the Google Play Store.
_ADocP_Dir	numeric	Sets the get-directory permission if > 0 within the ADoc Provider. Default is 0. ADocProviderDir.txt returns the updated directory content. ADocProviderDateDir.txt returns the updated directory content with a timestamp at the begin of each record. If it is 0 both files return an empty string. Note, that _ADocP_Read have to be > 0 also.
_ADocP_DirOut	numeric	If > 0, directories will be returned also. Default = 0
_ADoc_Summary	String	Summary text under app name and icon. Default is "".

File Providers support only read and write files. OliBasic supports also the recursive directory content by reading out FileProviderDir.txt and FileProviderDateDir.txt.

The file path of the File Provider is:

```
Program.info bInf
Bundle.get bInf, "_PackageName", pn$
"content://" + pn$ + "/" + yourFilePath$
```

Note: yourFilePath$ is the path within the File Provider directory.

ADoc Providers support the bunch of the ADoc commands. OliBasic supports also the recursive directory content by reading out ADoc ProviderDir.txt and ADoc ProviderDateDir.txt.

The root directory of the Adoc Provider is:

```
Program.info bInf
Bundle.get bInf, "_PackageName", pn$
"content://" + pn$ + ".documents/document/"+ "root" + "%3A"  + yourDocumentPath$
```

Note: yourDocumentPath$ is the path within the Adoc Provider directory.

To get the content of a File provided by a File Provider use:

```
Adoc.read success, result$, "content://com.my.app" + "/" + "test.txt", , 0
```

In case of OliBasic itself:

```
File.root iPath$, "_FileProvider"
Byte.open w, ftp, iPath$ + "/" + "testText.txt"
Byte.write.buffer ftp, "Basic4ever\nFile Provider"
Byte.close ftp
Adoc.read success, result$, "content://com.rfo.basicOli" + "/" + "testText.txt",
```

```
, 0
Print success, result$
```

For your compiled app, replace the package name from OliBasic to yours, for example:

```
"content://com.rfo.basicOli" into "content://com.my.newapp".
```

48 QR Code

48.1 QR.create.svg

Syntax: QR.create.svg <fileName_sexp>, <text_sexp>{{, <level_sexp>}, <bundlePtr_nvar>}

Creates an SVG vector file containing a QR code. The vector file name is defined by <fileName_sexp>. The QR code will be created from the text given by <text_sexp>.

The level of error correction will be set optionally by <level_sexp>. Default is "_Medium".

Level L ("_Low")	7% of data bytes can be restored.
Level M ("_Medium")	15% of data bytes can be restored.
Level Q ("_Quartile")	25% of data bytes can be restored.
Level H ("_High")	30% of data bytes can be restored.

Optional bundle <bundlePtr_nvar> content:

Table of SVG control options		
Key	**Value**	**Description**
_Border	numeric	Sets the border around the QR code. Default is 1.
_PatternColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	The Alpha channel will be not interpreted in opposite to _BackgroundColor.
_BackgroundColor	{Alpha,} Red, Green, Blue (comma delimited string) or _{Alpha,} ColorName ({comma delimited} string) or #{hn}hnhnhn (hex string)	
_Size	String	Sets the size of the quadratic QR code. The measurement units can be nothing, pxl, in, cm or mm. If the string is empty, the width and height parameters of the SVG header will be deleted. In this case Webview and HTML browsers scale the SVG file within the display borders. But this file cannot be load by OliBasic because it needs width and height. Example: "50mm"

Example:

```
! Confugration of the QR code result
Bundle.put b, "_Border", 3
Bundle.put b, "_PatternColor", "_DarkRed"
Bundle.put b, "_BackgroundColor", "_LightPink"
Bundle.put b, "_Size", "19cm"
```

```
Gr.open "_Gray", 1, 1
Gr.screen mGrWidth, mGrHeight
Qr.create.svg "test.svg", "Hello World!", "_High", b
Gr.bitmap.load bmpPtr, "test.svg", mGrWidth,  mGrHeight

Gr.statusbar inset, w
Gr.bitmap.draw oPtr1, bmpPtr, 0, inset
Gr.render

Do
 Pause 100
Until 0
End
```

49 Random Number Generator

49.1 Randomize

Syntax: Randomize({<nexp>})

Creates a pseudo-random number generator for use with the **Rnd()** function. The optional seed parameter <nexp> initializes the generator. Omitting the parameter is the same as specifying 0. If you call **Rnd()** without first calling **Randomize()**, it is the same as if you had executed **Randomize(0)**.

A non-zero seed initializes a predictable series of pseudo-random numbers. That is, for a given non-zero seed value, subsequent **Rnd()** calls will always return the same series of values.

If the seed is 0, the sequence of numbers from **Rnd()** is unpredictable and not reproducible. However, repeated **Randomize(0)** calls do not produce "more random" sequences.

The **Randomize()** function always returns zero.

49.2 Rnd

Syntax: Rnd()

Returns a random number generated by the pseudorandom number generator. If a **Randomize(n)** has not been previously executed then a new random generator will be created using **Randomize(0)**.

The random number will be greater than or equal to zero and less than one. (0 <= n < 1).

```
d = Floor(6 * Rnd() + 1)        % roll a six-sided die
```

50 Read Commands

50.1 Read.data

Syntax: Read.data <number>|<string>{,<number>|<string>...,<number>|<string>}

Provides the data value(s) to be read with **Read.next**.

Read.data statements may appear anywhere in the program. You may have as many **Read.data** statements as you need.

Example:

```
Read.data 1,2,3,"a","b","c"
```

Read.data is equivalent to the **DATA** statement in Dartmouth Basic.

50.2 Read.from

Syntax: Read.from <nexp>

Sets the internal NEXT pointer to the value of the expression. This command can be set to randomly access the data.

The command **Read.from 1** is equivalent to the **RESTORE** command in Dartmouth Basic.

50.3 Read.next

Syntax: Read.next <var>, ...

Reads the data pointed to by the internal NEXT pointer into the next variables. The NEXT pointer is initialized to "1" and is incremented by one each time a new value is read. Data values are read in the sequence in which they appeared in the program **Read.data** statement(s).

The data type (number or string) of the variable must match the data type pointed by the NEXT pointer.

Example:

```
Read.next a,b,c,c$
Read.next d$,e$
```

Read.next is equivalent to the **READ** statement in Dartmouth Basic.

51 Ringer Commands

Android devices support three ringtone modes:

Value:	Meaning:	Behavior
0	Silent	Ringer is silent and does not vibrate
1	Vibrate	Ringer is silent but vibrates
2	Normal	Ringer may be audible and may vibrate

"Normal" behavior depends on other device settings set by the user.

Ringer volume is an integer number from zero to a device-dependent maximum. If the volume is zero the ringer is silent.

NOTE: These are system settings. Any change you make persists after your program ends. You may want to record the original settings and change them back when the program exits.

51.1 Ringer.get.mode

Syntax: Ringer.get.mode <nvar>

Returns the current ringtone mode in the numeric variable.

51.2 Ringer.get.volume

Syntax: Ringer.get.volume <vol_nvar> { , <max_nvar> }

Returns the ringer volume level in the numeric variable. Returns the maximum volume settting in <max_nvar>, if <max_nvar> is present.

51.3 Ringer.set.mode

Syntax: Ringer.set.mode <nexp>

Changes the ringtone mode to the specified value. If the value is not a valid mode, the device mode is not changed.

51.4 Ringer.set.volume

Syntax: Ringer.set.volume <nexp>

Changes the ringer volume to the specified value. If the value is less than zero, volume is set to zero. If the value is greater than the device-specific maximum, the volume is set to the maximum level.

52 Sensors Commands

Android devices can have several types of sensors. A lot of programming ideas are based on sensors, so sensors are important. Each sensor can return up to 16 parameters. Currently, Android supports defined sensor types from 1 to 42. However, an Android device may have other, non standard sensors created by its vendor. These sensors have type numbers greater than 65535. These sensors always return 16 parameters.

Unused parameters always return 0.0.

Some sensors need user permissions. If so, the user will be asked to grant permission.

Obviously, not all Android devices have all of sensors. The command **Sensors.list** can be used to provide an inventory of the sensors available on a particular device. And to prevent a runtime error, use **Sensors.exists**, because you never know which sensors are available in another user's device.

A special note about sensor type 3:

The popular sensor type (+)3 has been deprecated since Android 4.4 (API 20). A special type of -3 has been developed a replacement. The sensor commands do not normally allow a negative type, but will treat this request as a special case. Sensor type -3 uses the Acceleration and Magnetic Field sensors. This pseudo sensor follows this guideline:

https://developer.android.com/guide/topics/sensors/sensors_position?hl=en#sensors-pos-orient

Also, the old sensor type 3 had the roll parameter reversed, this one does not. So this fix is in accord with the guideline.

Example:

```
Sensors.open -3
Sensors.read -3, azimuth, pitch, roll
```

52.1 Sensors.close

Syntax: Sensors.close

Closes the previously opened sensors. The sensors' hardware will be turned off preventing battery drain. Sensors are automatically closed when the program run is stopped via the BACK key.

52.2 Sensors.exists

Syntax: Sensors.exists <exists_nvar>, <sensor_type_nexp>

This command returns 1 by <exists_nvar> if the sensor type specified by <sensor_type_nexp> exists on the current device otherwise it returns 0.

52.3 Sensors.list

Syntax: Sensors.list <sensor_array$[]>{, <all_nexp>}

Writes information about the sensors available on the Android device into the <sensor_array$[]> parameter. If the array exists, it is overwritten. Otherwise a new array is created. The result is always a one-dimensional array.

The array elements contain the names and types of the available sensors. For example, one element may be "Gyroscope, Type = 4".

If <all_nexp> is set to 1, all available information will be presented. This only works for Android 5+. By default, <all_nexp> is set to 0.

The following program snippet prints the elements of the sensor list.

```
Sensors.list sensorarray$[]
Array.row.print sensorarray$[]
```

52.4 Sensors.open

Syntax: Sensors.open <type_nexp>{:<delay_nexp>}{, <type_nexp>{:<delay_nexp>}, ...}

Opens a list of sensors for reading. The parameter list is the type numbers of the sensors to be opened, followed optionally by a colon (':') and a number (0, 1, 2, or 3) that specifies the delay in sampling the sensor. 3 is the default (slowest).

This table gives a general idea of what the rate values mean. The delay values are only "suggestions" to the sensors, which may alter the real delays, and do not apply to all sensors. Faster settings use more battery.

Value	Name	Typical Delay	Typical Usage
3	Normal	200 ms	Default: suitable for screen orientation changes
2	UI	60 ms	Rate suitable for the user interface
1	Game	20 ms	Rate suitable for game play
0	Fastest	0 ms	Sample as fast as possible

Example:

```
Sensors.open 1:1, 3  % Monitor the Acceleration sensor at Game rate
                     % and the Orientation sensor at Normal rate.
```

This command must be executed before issuing any **Sensors.read** commands. You should only open the sensors that you actually want to read. Each sensor opened increases battery drain and the background CPU usage.

BASIC! uses the colon character to separate multiple commands on a single line. The use of colon in this command conflicts with that feature, so you must use caution when using both features together.

If you put any colons on a line after this command, the preprocessor **always** assumes the colons are part of the command and not command separators. The **Sensors.open** command **must** be either on a line by itself or placed last on a multi-command line.

52.5 Sensors.read

Syntax: Sensors.read <sensor_type_nexp>, <p1_nvar>, <p2_nvar>, <p3_nvar> {, <param_array[]>}

This command returns that latest values from the sensors specified by the "sensor_type" parameters. The returned values are placed in the <p1_nvar>, <p2_nvar> and <p3_nvar> parameters. The meaning of these parameters depends upon the sensor being read. Not all sensors return all three parameter values. In such cases, the unused parameters will be set to zero.

The optional <param_array[]> returns an array with three or more parameters. All available sensor types, and up to 16 parameters are supported. It is recommended to use <param_array[]>, because many sensors return more than 3 parameters.

See also:

https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/hardware/Sensor.java

53 Socket (TCP/IP) Commands

TCP/IP Sockets provide for the transfer of information from one point on the Internet to another. There are two genders of TCP/IP Sockets: Servers and Clients. Clients must talk to Servers. Servers must talk to Clients. Clients cannot talk to Clients. Servers cannot talk to Servers.

Every Client and Server pair have an agreed-upon protocol. This protocol determines who speaks first and the meaning and sequence of the messages that flow between them.

Most people who use a TCP/IP Socket will use a Client Socket to exchange messages with an existing Server with a predefined protocol. One simple example of this is the Sample Program file, **f31_socket_time.bas**. This program uses a TCP/IP client socket to get the current time from one of the many time servers in the USA.

A TCP/IP Server can be set up in BASIC!; however, there are difficulties. The capabilities of individual wireless networks vary. Some wireless networks allow servers. Most do not. Servers can usually be run on WiFi or Ethernet Local Area Networks (LAN).

If you want to set up a Server, the way most likely to work is to establish the Server inside a LAN. You will need to provide Port tunneling (forwarding) from the LAN's external Internal IP to the device's LAN IP. You must to be able to program (setup) the LAN router in order to do this.

Clients, whether running inside the Server's LAN or from the Internet, should connect to the LAN's external IP address using the pre-established, tunneled Port. This external or WAN IP can be found using:

```
GrabUrl ip$, "http://icanhazip.com"
```

This is not the same IP that would be obtained by executing **Socket.myIP** on the server device.

Note: The specified IPs do not have to be in the numeric form. They can be in the name form.

The Sample Program, **f32_tcp_ip_sockets.bas**, demonstrates the socket commands for a Server working in conjunction with a Client. You will need two Android devices to run this program.

On Android devices the default transfer character set is the files system character set UTF-8. If you need to transfer characters from 0 to 255, maybe for binary data, you can choose the "_ISO-8859-1" character set instead of the default "_UTF-8". But the commands **Socket.server.write.file**, **Socket.server.read.file**, **Socket.client.write.file** and **Socket.client.read.file** do not work as expected with "_ISO-8859-1" because sending and receiving of **Chr$**(65535) is not possible.

An ephemeral port is a communications endpoint (port) of a transport layer protocol of the Internet protocol suite that is used for only a short period of time (the duration of a communication session). Such short-lived ports are allocated automatically within a predefined range of port numbers by the IP stack software of a computer operating system.

How to choose one? The RFC 6056 says that the range for ephemeral ports should be 1024–65535. IANA and RFC 6335 suggests the range 49152–65535 for dynamic or private ports. Older Windows versions and BSD use ports 1025–5000 as ephemeral ports. Many Linux kernels use the port range 32768–60999. Android use a Linux kernel, thus this port range is recommended.

Four-digit ports such as 2021 with the ending 21 (FTP) or 8080 with the ending 80 (HTTP) are also common for permanent connections.

53.1 Socket Client (TCP/IP) Commands

53.1.1 Socket.client.close

Syntax: Socket.client.close

Closes an open client side connection.

53.1.2 Socket.client.connect

Syntax: Socket.client.connect <server_sexp>, <port_nexp> {{ , <wait_lexp> }, <char_set_sexp>}

Create a Client TCP/IP socket and attempt to connect to the Server whose Host Name or IP Address is specified by the Server string expression using the Port specified by Port numeric expression.

The optional "wait" parameter determines if this command waits until a connection is made with the Server. If the parameter is absent or true (non-zero), the command will not return until the connection has been made or an error is detected. If the Server does not respond, the command should time out after a couple of minutes, but this is not certain.

If the parameter is false (zero), the command completes immediately. Use **Socket.client.status** to determine when the connection is made. If you monitor the socket status, you can set your own time-out policy. You must use the **Socket.client.close** command to stop a connection attempt that has not completed.

With the optional <charset_sexp>, you can choose between following character sets: "_ISO-8859-1" and "_UTF-8"(default).

53.1.3 Socket.client.read.byte

Syntax: Socket.client.read.byte <svar>

Read a byte from the previously connected Server, and place the byte into the string variable. To avoid an infinite delay while waiting for the Server to send a line, the **Socket.client.read.ready** command can be repeatedly executed with timeouts.

What About Reading Unknown Number of Bytes?

The best answer is that your application either needs to know beforehand how many bytes to expect, or the "application protocol" needs to somehow tell it how many bytes to expect, or when all bytes have been sent. Possible approaches are:

1. The application protocol uses fixed message sizes
2. The application protocol message sizes are specified in message headers
3. The application protocol uses end-of-message markers
4. The application protocol is not message based, and the other end closes the connection to say "that's the end".

```
line$ = ""
Do
 Socket.client.read.byte mByte$
 line$ = line$ + mByte$
Until mByte$ = Chr$(10) | mByte$ = Chr$(13)
line$ = Replace$(line$, Chr$(10), "")
line$ = Replace$(line$, Chr$(13), "")
```

Is the same as

```
Socket.client.read.line line$
```

```
byteAsNumber = 0 ... 65535
out$ = Chr$(byteAsNumber )
Socket.server.write.bytes out$

Socket.client.read.byte in$
byteAsNumber = Ucode(in$)
```

53.1.4 Socket.client.read.file

Syntax: Socket.client.read.file <file_nexp>

Read file data transmitted by the Server and write it to a file. The <file_nexp> is the file index of a file opened for write by **Byte.open write** command. For example:

```
Byte.open w, fw, "image.jpg"
Socket.client.read.file fw
Byte.close fw
```

53.1.5 Socket.client.read.line

Syntax: Socket.client.read.line <line_svar>

Read a line from the previously-connected Server and place the line into the line string variable. The command does not return until the Server sends a line. To avoid an infinite delay waiting for the Server to send a line, the **Socket.client.read.ready** command can be repeatedly executed with timeouts.

Note: The end of the line is detected by receiving a CR or a LF.

53.1.6 Socket.client.read.ready

Syntax: Socket.client.read.ready <nvar>

If the previously created Client socket has not received a line for reading by **Socket.client.read.line** then set the return variable <nvar> to zero. Otherwise return a non-zero value.

The **Socket.client.read.line** command does not return until a line has been received from the Server. This command can be used to allow your program to time out if a line has not been received within a pre-determined time span. You can be sure that **Socket.client.read.line** will return with a line of data if **Socket.client.read.ready** returns a non-zero value.

53.1.7 Socket.client.server.ip

Syntax: Socket.client.server.ip <svar>

Return the IP of the server that this client is connected to in the string variable.

53.1.8 Socket.client.status

Syntax: Socket.client.status <status_nvar>

Get the current client socket connection status and place the value in the numeric variable <status_nvar>.

0 = Nothing going on
2 = Connecting
3 = Connected

53.1.9 Socket.client.write.bytes

Syntax: Socket.client.write.bytes <sexp>

Send the string expression, <sexp>, to the previously-connected Server as 8-bit bytes. Each character

of the string is sent as a single byte. The string is not encoded. No end-of-line characters are added by BASIC!. If you need a CR or LF character, you must make it part of the string. Note that if **Socket.server.read.line** is used to receive these bytes, the **read.line** command will not return until it receives a LF (10, 0x0A) character.

53.1.10 Socket.client.write.file

Syntax: Socket.client.write.file <file_nexp>

Transmit a file to the Server. The <file_nexp> is the file index of a file opened for read by **Byte.open**. Example:

```
Byte.open r, fr, "image.jpg"
Socket.client.write.file fr
Byte.close fr
```

53.1.11 Socket.client.write.line

Syntax: Socket.client.write.line <line_sexp>

Send the string expression <line_sexp> to the previously-connected Server as UTF-16 characters. End of line characters will be added to the end of the line.

53.2 Socket Server (TCP/IP) Commands

53.2.1 Socket.myIP

Syntax: Socket.myIP <svar>

Returns the IP of the device in string variable <svar>. If the device has no active IP address, the returned value is the empty string "".

If the device is on a WiFi or Ethernet LAN then the IP returned is the device's LAN IP.

Note: The external or WAN IP can be found using:

```
GrabUrl ip$, "http://icanhazip.com"
```

53.2.2 Socket.myIP

Syntax: Socket.myIP <array$[]>{, <nvar>}

Returns all active IP addresses of the device in the string array <array$[]>. If you provide the optional address-count variable <nvar>, it is set to the number of active IP addresses.

If the device has no active IP address, the array has a single element, the empty string "", and the address-count in <nvar> is 0. In this case only, the address-count is not the same as the array length.

Most devices usually have zero or one IP address. It is possible to have more than one. For example, after enabling a WiFi connection, there may still be an active cellular data connection. Normally this connection shuts down after a short time, but in some cases it may remain open.

53.2.3 Socket.server.client.ip

Syntax: Socket.server.client.ip <nvar>

Return the IP of the Client currently connected to the Server.

53.2.4 Socket.server.close

Syntax: Socket.server.close

Close the previously created Server. Any currently connected client will be disconnected.

53.2.5 Socket.server.connect

Syntax: Socket.server.connect {<wait_lexp>}

Direct the previously created Server to accept a connection from the next client in the queue.

The optional "wait" parameter determines if the command waits until a connection is made with a client. If the parameter is absent or true (non-zero), the command waits for the connection. If the parameter is false (zero), the command completes immediately. Use **Socket.server.status** to determine when the connection is made.

In general, it is safer to set the parameter to false (don't wait) and explicitly monitor the connection's status, since it can avoid a problem if the program exits with no connection made. You must use the **Socket.server.close** command to stop a connection attempt that has not completed.

53.2.6 Socket.server.create

Syntax: Socket.server.create <port_nexp>{, char_set_sexp}

Establish a Server that will listen to the Port specified by the numeric expression, <port_nexp>. With the optional <charset_sexp> you can choose between following character sets: "_ISO-8859-1" and "_UTF-8"(default).

53.2.7 Socket.server.disconnect

Syntax: Socket.server.disconnect

Close the connection with the previously-connected Client. A new **Socket.server.connect** can then be executed to connect to the next client in the queue.

53.2.8 Socket.server.read.byte

Syntax: Socket.server.read.byte <svar>

Read a byte sent from the previously connected Client and place the byte into the string variable <svar>. To avoid an infinite delay while waiting for the Client to send a line, the **Socket.server.read.ready** command can be repeatedly executed with timeouts.

53.2.9 Socket.server.read.file

Syntax: Socket.server.read.file <file_nexp>

Read file data transmitted by the Client and write it to a file. The <file_nexp> is the file index of a file opened for write by **Byte.open**. Example:

```
Byte.open w, fw, "image.jpg"
Socket.server.read.file fw
Byte.close fw
```

53.2.10 Socket.server.read.line

Syntax: Socket.server.read.line <svar>

Read a line sent from the previously-connected Client and place the line into the string variable <svar>. The command does not return until the Client sends a line. To avoid an infinite delay waiting for the

Client to send a line, the **Socket.server.read.ready** command can be repeatedly executed with timeouts.

Note: The end of the line is detected by receiving a CR or a LF.

53.2.11 Socket.server.read.ready

Syntax: Socket.server.read.ready <nvar>

If the previously-connected Client socket has not sent a line for reading by **Socket.server.read.line** then set the return variable <nvar> to zero. Otherwise return a non-zero value.

The **Socket.server.read.line** command does not return until a line has been received from the Client. This command can be used to allow your program to time out if a line has not been received within a pre-determined time span. You can be sure that **Socket.server.read.line** will return with a line of data if returns a non-zero value.

53.2.12 Socket.server.status

Syntax: Socket.server.status <status_nvar>

Get the current server socket connection status and place the value in the numeric variable <status_nvar>.

- 1 = Server socket not created
- 0 = Nothing going on
- 1 = Listening
- 3 = Connected

53.2.13 Socket.server.write.bytes

Syntax: Socket.server.write.bytes <sexp>

Send the string expression, <sexp>, to the previously-connected Client as 8-bit bytes. Each character of the string is sent as a single byte. The string is not encoded. No end of line characters are added by BASIC!. If you need a CR or LF character, you must make it part of the string. Note that if **Socket.client.read.line** is used to receive these bytes, the **read.line** command will not return until it receives a LF (10, 0x0A) character.

53.2.14 Socket.server.write.file

Syntax: Socket.server.write.file <file_nexp>

Transmit a file to the Client. The <file_nexp> is the file index of a file opened for read by **Byte.open**. Example:

```
Byte.open r, fr, "image.jpg"
Socket.server.write.file fr
Byte.close fr
```

53.2.15 Socket.server.write.line

Syntax: Socket.server.write.line <line_sexp>

Send the string expression <line_sexp> to the previously-connected Client as UTF-16 characters. End of line characters will be added to the end of the line.

54 Socket (UDP) Comands

User Datagram Protocol (UDP) is a simpler message-based connectionless protocol. Connectionless protocols do not set up a dedicated end-to-end connection. Communication is achieved by transmitting information in one direction from source to destination without verifying the readiness or state of the receiver.

Some characteristics are:

Unreliable – When an UDP message is sent, it cannot be known if it will reach its destination; it could get lost along the way. There is no concept of acknowledgment, retransmission, or timeout.

Not ordered – If two messages are sent to the same recipient, the order in which they arrive cannot be predicted.

Lightweight – There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.

Datagrams – Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning that a read operation at the receiver socket will yield an entire message as it was originally sent.

No congestion control – UDP itself does not avoid congestion. Congestion control measures must be implemented at the application level.

Broadcasts – being connectionless, UDP can broadcast - sent packets can be addressed to be receivable by all devices on the subnet.

Port range – from 32768 to 60999 is recommended.

A transfer request is always initiated targeting port 69, but the data transfer ports are chosen independently by the sender and receiver during the transfer initialization.

If you want a read and write communication (IoT), make sure that there is a pause of maybe 500ms for interaction.

54.1 UDP.read

Syntax: UDP.read <result_svar>, <port_nexp>, <wait_nexp> {, <char_set_sexp>}

Listens for an UDP message on port <port_nexp> and reads a string if any datagram message arrived. The command waits <wait_nexp> milliseconds. If <wait_nexp> is 0 it waits forever. Using this command in a loop with 1200 ms for the first try is recommended. The character set can be specified by <char_set_sexp>. Use "_UTF-8" (default) for text and "_ISO-8859-1" for binary data. An input until 65527 bytes is accepted.

Example:

```
Do
 Do
  Udp.read r$, 54321, 1200, "_ISO-8859-1"
 Until r$ <> ""
 Print r$
Until 0
```

54.2 UDP.write

Syntax: UDP.write <message_svar>, <ip_adress_sexp>, <port_nexp> {, <char_set_sexp>}

Writes a string <message_svar> as an UDP datagram message to a local IP adress <ip_adress_sexp> on port <port_nexp>. The character set can be specified by <char_set_sexp>. Use "_UTF-8" (default) for text and "_ISO-8859-1" for binary data. An output until 65507 bytes is accepted.

But keep in mind that large messages will be split.

Example:

```
m$= "My message! "
For i = 1 To 100
 mm$ = m$ + Str$(i)
 Udp.write mm$, "192.168.1.12", 54321, "_ISO-8859-1"
 Pause 50
Next
```

55 SoundPool Commands

A SoundPool is a collection of short sound bites that are preloaded and ready for instantaneous play. SoundPool sound bites can be played while other sounds are playing, either while other sound bites are playing or over a currently playing sound file being played my means of **Audio.play**. In a game, the **Audio.play** file would be the background music while the SoundPool sound bites would be the game sounds (Bang, Pow, Screech, etc).

A SoundPool is opened using the **SoundPool.open** command. After the SoundPool is opened, sound bites will be loaded into memory from files using the **SoundPool.load** command. Loaded sound bites can be played over and over again using the **SoundPool.play** command.

A playing sound is called a sound stream. Individual sound streams can be paused with **SoundPool.pause**, individually or as a group, resumed with **SoundPool.resume** and stopped with **SoundPool.stop**. Other stream parameters (priority, volume and rate) can be changed on the fly.

The **SoundPool.release** command closes the SoundPool. A new SoundPool can then be opened for a different phase of the game. **SoundPool.release** is automatically called when the program run is terminated.

55.1 SoundPool.load

Syntax: SoundPool.load <soundID_nvar>, <file_path_sexp>

The file specified in <file_path_sexp> is loaded. Its sound ID is returned in <soundID_nvar>. The sound ID is used to play the sound and also to unload the sound. The sound ID will be returned as zero if the file was not loaded for some reason.

The default file path is "sdcard/rfo-basic/data/"

Note: It can take a few hundred milliseconds for the sound to be loaded. Insert a "Pause 500" statement after the load if you want to play the sound immediately following the load command.

55.2 SoundPool.open

Syntax: SoundPool.open <MaxStreams_nexp>

The MaxStreams expression specifies the number of Soundpool streams that can be played at once. If the number of streams to be played exceeds this value, the lowest priority streams will be terminated.

Note: A stream playing via audio.play is not counted as a Soundpool stream.

55.3 SoundPool.pause

Syntax: SoundPool.pause <streamID_nexp>

Pauses the playing of the specified stream. If the stream ID is zero, all streams will be paused.

55.4 SoundPool.play

Syntax: SoundPool.play <streamID_nvar>, <soundID_nexp>, <rightVolume_nexp>, <leftVolume_nexp>, <priority_nexp>, <loop_nexp>, <rate_nexp>

Starts the specified sound ID playing.

The stream ID is returned in <streamID_nvar>. If the stream was not started, the value returned will be zero. The stream ID is used to pause, resume and stop the stream. It is also used in the stream

modification commands (**SoundPool.setRate**, **SoundPool.setVolume**, **SoundPool.setPriority** and **SoundPool.setLoop**).

The left and right volume values must be in the range of 0 to 0.99 with zero being silent.

The priority is a positive value or zero. The lowest priority is zero.

The loop value of -1 will loop the playing stream forever. Values other than -1 specify the number of times the stream will be replayed. A value of 1 will play the stream twice.

The rate value changes the playback rate of the playing stream. The normal rate is 1. The minimum rate (slow) is 0.5. The maximum rate (fast) is 1.85.

55.5 SoundPool.release

Syntax: SoundPool.release

Closes the SoundPool and releases all resources. **SoundPool.open** can be called to open a new SoundPool.

55.6 SoundPool.resume

Syntax: SoundPool.resume <streamID_nexp>

Resumes the playing of the specified stream. If the stream ID is zero, all streams will be resumed.

55.7 SoundPool.setPriority

Syntax: SoundPool.setPriority <streamID_nexp>, <priority_nexp>

Changes the priority of a playing stream.

The lowest priority is zero.

55.8 SoundPool.setRate

Syntax: SoundPool.setRate <streamID_nexp>, <rate_nexp>

Changes the playback rate of the playing stream.

The normal rate is 1. The minimum rate (slow) is 0.5. The maximum rate (fast) is 1.85.

55.9 SoundPool.setVolume

Syntax: SoundPool.setVolume <streamID_nexp>, <leftVolume_nexp>, <rightVolume_nexp>

Changes the volume of a playing stream.

The left and right volume values must be in the range of 0 to 0.99 with zero being silent.

55.10 SoundPool.stop

Syntax: SoundPool.stop <streamID_nexp>

Stops the playing of the specified stream.

55.11 SoundPool.unload

Syntax: SoundPool.unload <soundID_nexp>

The specified loaded sound is unloaded.

56 Speech Conversion

56.1 Text To Speech

Your program can synthesize speech from text, either for immediate playback with the **TTS.speak** command or to create a sound file with **TTS.speak.toFile**.

Your device may come with the text-to-speech engine already enabled and configured, or you may need to set it up yourself in the Android Settings application. The details vary between different devices and versions of Android. Typically, the menu navigation looks like one of these:

Settings → Voice input & output → Text-to-speech settings
Settings → Language & input → Text-to-speech output

Unless you set an output language in the text-to-speech settings, the speech generated will be spoken in the current default language of the device. The menu path for setting the default language usually looks like one of these:

Settings → Language and keyboard → Select language
Settings → Language & input → Language

Most speech engines limit the number of characters they are able to speak. The limit is not the same for all devices or all speech engines, but it is typically around 4000 characters. If you exceed the limit, most engines fail silently: you don't get an error message, but you don't get any speech output, either.

56.1.1 TTS.init

Syntax: TTS.init

This command must be executed before speaking.

56.1.2 TTS.kill

Syntax: TTS.kill

TTS.kill stops the TTS process unconditionally. Be careful with this command! You have to make sure that all following commands, including **TTS.stop**, will not be executed. Following **TTS.kill**, if you want to run **TTS.speak** or **TTS.speak.toFile** again, you will have to first run **TTS.init** again.

56.1.3 TTS.speak

Syntax: TTS.speak <sexp> {, <wait_lexp>}

Speaks the string expression. The statement does not return until the string has been fully spoken, unless the optional "wait" parameter is present and evaluates to false (numeric 0). Spoken expressions cannot overlap. A second **TTS.speak** (or a **TTS.speak.toFile**) will wait for the speech from an earlier **TTSts.speak** to finish, even if the "wait" flag was false.

56.1.4 TTS.speak.toFile

Syntax: TTS.speak.toFile <sexp> {, <path_sexp>}

Converts the string expression to speech and writes it into a wav file. You can specify the name and location of the file with the optional "path" parameter. The default path is "<pref base drive>/rfo-basic/data/tts.wav". The statement does not return until the speech synthesis is complete, but there is no guarantee the file-write is finished. If a previous **TTS.speak** is still speaking, this statement will not start until that speech completes.

56.1.5 TTS.stop

Syntax: TTS.stop

Waits for any outstanding speech to finish, then releases Android's text-to-speech engine. Following **TTS.stop**, if you want to run **TTS.speak** or **TTS.speak.toFile** again, you will have to run **TTS.init** again.

56.2 Speech To Text (Voice Recognition)

The Voice Recognition function on Android uses Google Servers to perform the recognition. This means that you must be connected to the Internet and logged into your Google account for this feature to work.

There are two commands for Speech to Text: **STT.listen** and **STT.results**.

STT.listen starts the voice recognition process with a dialog box. **STT.results** reports the interpretation of the voice with a list of strings.

The Speech to Text procedures are different for Graphics Mode, HTML mode and simple Console Output mode.

56.2.1 STT.listen

Syntax: STT.listen {{<prompt_sexp>}, <extras_bundle_nexp>}

Start the voice recognize process by displaying a "Speak Now" dialog box. The optional prompt string expression <prompt_sexp> sets the dialog box's prompt. If you do not provide the prompt parameter, the default prompt "BASIC! Speech To Text" is used.

Begin speaking when the dialog box appears.

The recognition will stop when there is a pause in the speaking.

STT.results should be executed next.

Note: **STT.listen** is *not* to be used in HTML mode.

The optional bundle <extras_bundle_nexp> controls more options:

Key	Value	Description
_Hidden	0* or 1 (numeric)	Opens the recognizer without any dialogues. If an error occurs only "Recognition Canceled" will be returned. This option takes only effect within Console Mode. Default is 0. In this case the recognizer is in front.
_Off	0* or 1 (numeric)	The Values are set, but a recognition will be not performed. An option in case of using GW-Lib within HTML Mode. Default is 0. In this case the recognizer will be performed.
_MaxResults	numeric	Optional limit on the maximum number of results to return. If omitted the

Key	Value	Description
		recognizer will choose how many results to return.
_Language	String	Optional IETF language tag, for example "en-US". This tag informs the recognizer.
_MinimumLength	numeric	The minimum length of an utterance. We will not stop recording before this amount of time. Note that it is extremely rare you'd want to specify this value. If you don't have a very good reason to change these, you should leave them as they are per default. Note also that certain values may cause undesired or unexpected results - use judiciously! Additionally, depending on the recognizer implementation, these values may have no effect.
_CompletSilence	numeric	The amount of time that it should take after we stop hearing speech to consider the input complete. Note that it is extremely rare you'd want to specify this value. If you don't have a very good reason to change these, you should leave them as they are per default. Note also that certain values may cause undesired or unexpected results - use judiciously! Additionally, depending on the recognizer implementation, these values may have no effect.
_PossiblyCompleteSilence	numeric	The amount of time that it should take after we stop hearing speech to consider the input possibly complete. This is used to prevent the end pointer cutting off during very short mid-speech pauses. Note that it is extremely rare you'd want to specify this value. If you don't have a very good reason to change these, you should leave them as they are per default. Note also that certain values may cause undesired or unexpected results - use judiciously! Additionally, depending on the recognizer implementation, these values may have no effect.
_WebSearch	0* 1 Web search and others 2 Only web search (numeric) [Does not work properly with Google Assistant]	Min. Jelly Bean 4.1 (API 16) Prompts the user for speech, send it through a speech recognizer, and either display a web search result or trigger another type of action based on the user's speech. For security reasons the mode falls back to normal recognition, if the screen

Key	Value	Description
		is off or the device is locked. If Google Assistant installed, it will be used. The problem is, Google Assistant can not be closed by voice.
_HandsFree	0* or 1 (numeric)	Min. Jelly Bean 4.1 (API 16) Launches a start of a HandsFreeApp in a mode that will prompt the user for speech without requiring the user's visual attention or touch input. This activity may be launched while device is locked in a secure mode. Special care is be taken to ensure that the voice actions that are performed while hands free cannot compromise the device's security. The called application can return a list, a String or a Number from type Double in the returned intent **extra** bundle. But your result is in each case only a list from type Sting. If you create with Bundle.out your result, you can use **only** a string or a number.
_Package	String	Min. Jelly Bean 4.1 (API 16) Only in conjunction with _HandsFree. Package name of the whished App. Example: "com.rfo.basicOli"
_Component	String	Min. Jelly Bean 4.1 (API 16) Only in conjunction with _HandsFree and _Package. Example: "com.rfo.basicOli.Basic"
_Data	String as URL	Min. Jelly Bean 4.1 (API 16) Only in conjunction with _HandsFree and _Package. Example: FILE.ROOT fp$, "_External" filePath$ = "file://" + fp$ + "rfo-basic/source" mUrl$ = filePath$ + "/" + "freeHandsDemo.bas"

If the following paragraph in the AndroidManifest.xml is accessible, the program is also a possible hands free receiver like Google Assistant. But the advantage is, the program can be closed by voice. If you create a normal APK from your program, the paragraph should be deleted.

```
<!-- vv 2017-11-25gt Comment it out if it should not be a hands free receiver, too.
-->
<intent-filter >
      <action android:name="android.speech.action.VOICE_SEARCH_HANDS_FREE" />
      <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
<!-- ^^ 2017-11-25gt -->
```

56.2.2 STT.results

Syntax: STT.results <string_list_ptr_nexp>

The command must not be executed until after a **STT.listen** is executed (unless in HTML mode).

The recognizer returns several variations of what it thinks it heard as a list of strings. The first string in the list is the best guess.

The strings are written into the list that <string_list_ptr_nexp> points to. The previous contents of the list are discarded. If the pointer does not specify a valid string list, and the expression is a numeric variable, a new list is created and the variable is set to point to the new list.

Examples:

Output Console

The following code illustrates the command in Output Console (not HTML mode and not Graphics mode):

```
Print "Starting Recognizer"
STT.listen
STT.results theList
List.size theList, theSize
For k = 1 To theSize
  List.get theList, k, theText$
  Print theText$
Next k
End
```

Graphics Mode

This command sequence is to be used in graphics mode. Graphics mode exists after **Gr.open** and before **Gr.close**. (Note: Graphics mode is temporarily exited after **Gr.front 0**. Use the Console Mode if you have called **Gr.front 0**).

The primary difference is that **Gr.render** *must* be called after **STT.listen** and before **STT.results**.

```
Print "Starting Recognizer"
STT.listen
Gr.render
STT.results theList
List.size theList, theSize
For k = 1 To theSize
  List.get theList, k, theText$
  Print theText$
Next k
End
```

HTML Mode

This command sequence is used while in HTML mode. HTML mode exists after **Html.open** and before **Html.close**.

The primary difference is that the **STT.listen** command is *not* used in HTML mode. The **STT.listen** function is performed by means of an HTML datalink sending back the string "STT". The sending of "STT" by means of the datalink causes the Speak Now dialog box to be displayed.

When the datalink "STT" string is received by the BASIC! program, the **STT.results** command can be executed normally as it will contain the recognized text.

The sample file, f37_html_demo.bas, along with the associated HTML file, htmlDemo1.html (located in "rfo-basic/data/") demonstrates the use of voice recognition in HTML mode.

57 Sql Commands

The Android operating system provides the ability to create, maintain and access SQLite databases. SQLite implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The full details about SQLite can be found at http://www.sqlite.org/.

Good sources are at http://www.sqlitetutorial.net and https://www.sqlite.org/faq.html#q2. There is also a tutorial which can be found at http://www.w3schools.com/sql/default.asp.

By default, database files will be created on the base drive in the directory, "<pref base drive>/rfo-basic/databases/ ".

57.1 Sql.ccl

Clears the global SQL Cursor List.

57.2 Sql.close

Syntax: Sql.close <DB_pointer_nvar>

Closes a previously opened database. <DB_pointer_nvar> will be set to zero. The variable may then be reused in another **Sql.open** command. You should always close an opened database when you are done with it. Not closing a database can reduce the amount of memory available on your Android device.

57.3 Sql.delete

Syntax: Sql.delete <DB_pointer_nvar>, <table_name_sexp>{,<where_sexp>{,<count_nvar>} }

From the named table of a previously opened database, delete rows selected by the conditions established by the Where string expression. The Count variable reports the number of rows deleted.

The formation of the Where string is exactly the same as described in the **Sql.query** command. Both Where and Count are optional. If the Where string is omitted all rows are deleted, and the Count variable must be omitted, too.

57.4 Sql.drop_table

Syntax: Sql.drop_table <DB_pointer_nvar>, <table_name_sexp>

The table named <table_name_sexp> in the opened database pointed to by <DB_pointer_nvar> will be dropped (deleted) from the database if the table exists.

57.5 Sql.exec

Syntax: Sql.exec <DB_pointer_nvar>, <command_sexp>

Execute ANY non-query SQL command string ("CREATE TABLE", "DELETE", "INSERT", etc.) using a previously opened database.

Example:

```
dbname$ = "example.db"
Sql.open DB_Ptr, dbname$

tbname$ = "Birthdates"
Sql.drop_table DB_Ptr, tbname$     % Delete table if exists.
```

```
Sql.drop_table DB_Ptr, "SavedData" % Delete table if exists.

tbname$ = "Birthdates"
mColumns$ = "FirstName TEXT, Name TEXT "

CommandString$ = "CREATE TABLE IF NOT EXISTS " + tbname$ + "( " ~
 + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "+ mColumns$ + " )"
Sql.exec DB_Ptr, CommandString$ % Create a table with two columns

CommandString$ = "ALTER TABLE "+tbname$+" ADD " + "Birthdate TEXT"
Sql.exec DB_Ptr, CommandString$ % Add a column

CommandString$ = "ALTER TABLE " + tbname$ + " RENAME TO " + "SavedData"
Sql.exec DB_Ptr, CommandString$ % Rename the table in "SavedData"

CommandString$ = "VACUUM" % Rebuild the database to smaller size.
Sql.exec DB_Ptr, CommandString$

Sql.insert DB_Ptr, "SavedData", "FirstName", "Mike", "Name", "Smith", ~
 "Birthdate", "2021-03-23"
Sql.insert DB_Ptr, "SavedData", "FirstName", "Lara", "Name", "Smith", ~
 "Birthdate", "2021-03-24"
```

57.6 Sql.insert

Syntax: Sql.insert <DB_pointer_nvar>, <table_name_sexp>, C1$, V1$, C2$, V2$, ..., CN$, VN$

or

Syntax: Sql.insert <DB_pointer_nvar>, <table_name_sexp>, <delim_row_sexp>

Inserts a new row of data columns and values into a table in a previously opened database.

The <table_name_sexp> is the name of the table into which the data is to be inserted. All newly inserted rows are inserted after the last, existing row of the table.

C1$, V1$, C2$, V2$, ..., CN$, VN$: The column name and value pairs for the new row. These parameters must be in pairs. The column names must match the column names used to create the table. Note that the values are all strings. When you need a numeric value for a column, use the BASIC! **Str$**(n) to convert the number into a string. You can also use the BASIC! **Format$**(pattern$, N) to create a formatted number for a value. (The Values-as-strings requirement is a BASIC! SQL Interface requirement, not a SQLite requirement. While SQLite, itself, stores all values as strings, it provides transparent conversions to other data types. I have chosen not to complicate the interface with access to these SQLite conversions since BASIC! provides its own conversion capabilities.)

The second syntax uses a <delim_row_sexp> string expression formatted as:

"_Delimiter:" + one delimiter character as a numeric string + ";" + C1$ + dC$ + C1$ + … + dC$ + CN$.

Using other delimiter characters is of little advantage. It is possible to program the number of columns.

Example:

```
delimN = Ucode("€")
dC$ = Chr$(delimN)
head$ = "_Delimiter:" + Int$(delimN) + ";"
row$ = head$ + c1$ + dC$ + fn$ + dC$ + c2$ + dC$ + ln$ + dC$ + c3$ + dC$ + ~
 ga$ + dC$ + c4$  + dC$ + pa$
Sql.insert DB_Ptr, tbname$, row$
```

57.7 Sql.new_table

Syntax: Sql.new_table <DB_pointer_nvar>, <table_name_sexp>, C1$, C2$, ..., CN$

or

Syntax: Sql.new_table <DB_pointer_nvar>, <table_name_sexp>, <delim_row_sexp>

A single database may contain many tables. A table is made of rows of data. A row of data consists of columns of values. Each value column has a column name associated with it.

This command creates a new table with the name <table_name_sexp> in the referenced opened database, but only if the table does not exist. The column names for that table are defined by the following: C1$, C2$, ..., CN$. At least one column name is required. You may create as many column names as you need.

BASIC! always adds a Row Index Column named "_id" to every table. The value in this Row Index Column is automatically incremented by one for each new row inserted. This gives each row in the table a unique identifier. This identifier can be used to connect information in one table to another table. For example, the _id value for customer information in a customer table can be used to link specific orders to specific customers in an outstanding order database.

The second syntax uses a <delim_row_sexp> string expression formatted as:

"_Delimiter:" + one delimiter character as a numeric string + ";" + C1$ + dC$ + C1$ + … + dC$ + CN$.

Using other delimiter characters is of little advantage. It is possible to program the number of columns.

Example:

```
delimN = Ucode("€")
dC$ = Chr$(delimN)
head$ = "_Delimiter:" + Int$(delimN) + ";"
columns$ = head$ + c1$ + dC$ + c2$ + dC$ + c3$ + dC$ + c4$
Sql.new_table DB_Ptr, tbname$, columns$
```

57.8 Sql.next

Syntax: Sql.next <done_lvar>, <cursor_nvar>{, <cv_svars>}

Using the Cursor generated by a previous Query command (**Sql.query** or **Sql.raw_query**), step to the next row of data returned by the Query and retrieve the data.

<done_lvar> is a Boolean variable that signals when the last row of the Query data has been read.

<cursor_nvar> is a numeric variable that holds the Cursor pointer returned by a Query command. You may have more than one Cursor open at a time.

<cv_svars> is an optional set of column value string variables that return data from the database to your program. The Cursor carries the values from the table columns listed in the Query. **Sql.next** retrieves one row from the Cursor as string values and writes them into the <cv_svars>. If any of your columns are numeric, you can use the BASIC! **Val(**str$**)** function to convert the strings to numbers.

When this command reads a row of data, it sets the Done flag <done_lvar> to false (0.0). If it finds no data to read, it changes the Done flag to true (1.0) and resets the cursor variable <cursor_nvar> to zero. The Cursor can no longer be used for **Sql.next** operations. The cursor variable may be used with another Cursor from a different Query.

In its simplest form, <cv_svars> is a comma-separated list of string variable names. Each variable

receives the data of one column. If there are more variables than columns, the excess variables are left unchanged. If there are more columns than variables, the excess data are discarded.

```
Sql.next done, cursor, cv1$, cv2$, cv3$   % get data from up to three columns
```

The last (or only) variable may be a string array name with no index(es):

```
Sql.next done, cursor, cv$[]              % get data from ALL available columns
```

The data from any column(s) that are not written to string variables are written into the array. If no column data are written to the array, the array has one element, an empty string "". If the variable names an array that already exists, it is overwritten.

If the last (or only) <cv_svars> variable is an array, you may also add (after another comma) a numeric variable <ncol_nvar>. This variable receives the total number of columns in the cursor. Note that this is not necessarily the same as the size of the array.

```
Sql.next done, cursor, cv$[], nCols       % report number of columns available
```

The full specification for this command, including the optional array and column count, is as follows:

```
Sql.next <done_lvar>, <cursor_nvar> {, svar}... {, array$[] {, <ncol_nvar>}}
```

57.9 Sql.open

Syntax: Sql.open <DB_pointer_nvar>, <DB_name_sexp>

Opens a database for access. If the database does not exist, it will be created.

<DB_pointer_nvar> is a pointer to the newly opened database. This value will be set by the sql.open command operation. <DB_pointer_nvar> is used in subsequent database commands and should not be altered.

<DB_name_sexp> is the filename used to hold this database. The base reference directory is "<pref base drive>/com.rfo.basic/databases/". If <DB_name_sexp> = ":memory:" then a temporary database will be created in memory.

Note: You may have more than one database opened at the same time. Each opened database must have its own, distinct pointer.

57.10 Sql.ping

Syntax: Sql.ping <result_nvar>, <DB_pointer_nvar> {, <table_name_sexp> {, <column_name_sexp>}}

Returns the size (number of rows) of a database or table to <result_nvar>.

The return code is as follows;

-1	The table name does not exist
-2	The table name exists but column does not exist in the table
≥0	The total number of tables in the database or rows in the table.

Examples:

```
Sql.ping r, db                    % ping only the database
Sql.ping r, db, "mytable"         % ping a table,
```

```
Sql.ping r, db, "mytable", "col1"  % ping a table and column
```

57.11 Sql.query

Syntax: Sql.query <cursor_nvar>, <DB_pointer_nvar>, <table_name_sexp>, <columns_sexp> {, <where_sexp> {, <order_sexp>} }

Queries a table of a previously-opened database for some specific data. The command returns a Cursor named <Cursor_nvar> to be used in stepping through Query results.

The <columns_sexp> is a string expression with a list of the names of the columns to be returned. The column names must be separated by commas. An example is Columns$ = "First_name, Last_name, Sex, Age". If you want to get the automatically incremented Row Index Column then include the "_id" column name in your column list. Columns may be listed in any order. The column order used in the query will be the order in which the rows are returned.

The optional <where_sexp> is an SQL expression string used to select which rows to return. In general, an SQL expression is of the form <Column Name> <operator> <Value>. For example, Where$ = "First_name = 'John' " Note that the Value must be contained in single quotes. Full details about the SQL expressions can be found at: http://www.sqlite.org/lang_expr.html. If the Where parameter is omitted, all rows will be returned.

The optional <order_sexp> specifies the order in which the rows are to be returned. It identifies the column upon which the output rows are to be sorted. It also specifies whether the rows are to be sorted in ascending (ASC) or descending (DESC) order. For example, Order$ = "Last_Name ASC" would return the rows sorted by Last_Name from A to Z. This will sort upper and lower case separately (case sensitive). For case insensitive sorts, replace the "ASC" or "DESC" (ascending or descending) part of the <order_sexp> argument with "COLLATE NOCASE ASC" OR "COLLATE NOCASE DESC" respectively.

"COLLATE UNICODE" sorts in Unicode order and "COLLATE LOCALIZED" in order of the current database language setting. If not specified by **Sql.set_locale** the current system language setting is used.

If the Order parameter is omitted, the rows are not sorted. If the Order parameter is present, the Where parameter must also be present. If you want to return all rows, just set Where$ = "".

57.12 Sql.query.length

Syntax: Sql.query.length <length_nvar>, <cursor_nvar>

Report the number of records returned by a previous Query command, Given the Cursor returned by a Query, the command writes the number of records into <length_nvar>. This command cannot be used after all of the data has been read.

57.13 Sql.query.position

Syntax: Sql.query.position <position_nvar>, <cursor_nvar>

Report the record number most recently read using the Cursor of a Query command. Given the Cursor returned by a Query, the command writes the position of the Cursor into <Position_nvar>. Before the first Next command, the Position is 0. It is incremented by each Next command. A Next command after the last row is read sets its Done variable to true and resets the Cursor to 0. The Cursor can no longer be used, and this command can no longer be used with that Cursor.

57.14 Sql.raw_query

Syntax: Sql.raw_query <cursor_nvar>, <DB_pointer_nvar>, <query_sexp>

Execute ANY SQL Query command using a previously opened database and return a Cursor for the results.

Example:

```
tbname$ = "SavedData"
sql$ = "SELECT sql FROM sqlite_master WHERE tbl_name = '" + tbname$ + ~
 "' AND type = 'table';"
Sql.raw_query mCursor, DB_Ptr, sql$
Sql.next xdone, mCursor, rawTableSql$
If rawTableSql$ = ""
 Print "No database table with this name found!"
Else
 Split.all a$[], rawTableSql$, "[()]"
 Split.all b$[], a$[2], ","
 Array.length al, b$[]
 Dim columns$[al-1], fTypes$[al-1]
 For i = 2 To al
  Split.all c$[], b$[i], " "
  columns$[i-1] = c$[2]
  fTypes$[i-1] = c$[3]
 Next
EndIf

index = 1
Join.all columns$[], fields$, ","
If index Then fields$ = "_id," + fields$
Sql.query cursor, DB_Ptr, tbname$, fields$
delim$ = ";" % Different delimiter
Join.all columns$[], fields$, delim$
If index Then fields$ = " " + delim$ + fields$
csvLines$ = fields$ + Chr$(10)
xdone = 0
Do
 Sql.next xdone, cursor, cv$[]
 !!b
 For n = 2 To al % Add -1 at the end, if the index is not used.
  If fTypes$[n-1] = "TEXT" Then cv$[n] = Chr$(34) + cv$[n] + Chr$(34)
 Next
 !!e
 Join.all cv$[], fields$, delim$
 If !xdone Then csvLines$ += fields$ + Chr$(10)
Until xdone
csvLines$ += Chr$(10)
csvLines$ = Replace$(csvLines$, Chr$(10) + Chr$(10), "") % Del. last LF
Print csvLines$
```

57.15 Sql.set_locale

Syntax: Sql.set_locale <DB_pointer_nvar>, <locale_sexp>

The <locale_sexp> specifies output based on language and region. The locale specifies the language and region with standardized codes. The <locale_sexp> is a string containing zero or more codes separated by underscores.

The function accepts up to three codes. The first must be a language code, such as "en", "de" or "ja".

The second must be a region or country code, such as "FR", "US", or "IN". Some language and country combinations can accept a third code, called the "variant code".

The function also accepts the standard three-letter codes and numeric codes for country or region. For

example, "fr_FR", "fr_FRA", and "fr_250" are all equivalent.

The locale take only effect, if "COLLATE LOCALIZED" is part of the **Sql.query** <order_sexp> statement. You may need to install your chosen language on the Android device first to get the full effect.

Example:

```
Sql.set_locale DB_Ptr, "de_CH"
```

57.16 Sql.update

Syntax: Sql.update <DB_pointer_nvar>, <table_name_sexp>, C1$, V1$, C2$, V2$,...,CN$, VN${: <where_sexp>}

or

Syntax: Sql.update <DB_pointer_nvar>, <table_name_sexp>, <delim_row_sexp> {: <where_sexp>}

In the named table of a previously opened database, change column values in specific rows selected by the Where$ parameter <where_sexp>. The C$,V$ parameters must be in pairs. The colon character terminates the C$,V$ list and must precede the Where$ in this command. The Where$ parameter and preceding colon are optional.

BASIC! also uses the colon character to separate multiple commands on a single line. The use of a colon in this command conflicts with that feature. Use caution when using both together.

If you put a colon on a line after this command, the preprocessor **always** assumes the colon is part of the command and not a command separator. If you are not certain of the outcome, the safest action is to put the **Sql.update** command on a line by itself, or at the end of a multi-command line.

The second syntax uses a <delim_row_sexp> string expression formatted as:

"_Delimiter:" + one delimiter character as a numeric string + ";" + C1$ + dC$ + C1$ + … + dC$ + CN$.

Using other delimiter characters is of little advantage. It is possible to program the number of columns.

Example:

```
delimN = Ucode("€")
dC$ = Chr$(delimN)
head$ = "_Delimiter:" + Int$(delimN) + ";"
Where$ = "first_name = 'Tamasin' AND last_name = 'Washington' "
! Sql.update DB_Ptr, tbname$, c3$, "94": Where$ or
row$ = head$ + c3$ + de$ + "94"
Sql.update DB_Ptr, tbname$, row$: Where$
```

58 Stack Commands

Stacks are like a magazine for a gun. The last bullet into the magazine is the first bullet out of the magazine. This is also what is true about stacks. The last object placed into the stack is the first object out of the stack. This is called LIFO (Last In First Out).

An example of the use of a stack is the BASIC! **GoSub** command. When a **GoSub** command is executed the line number to return to is "pushed" onto a stack. When a **Return** is executed the return line number is "popped" off of the stack. This methodology allows **GoSub**s to be nested to any level. Any **Return** statement will always return to the line after the last **GoSub** executed.

A running example of Stacks can be found in the Sample Program file, **f29_stack.bas**.

There is no fixed limit on the size or number of stacks. You are limited only by the memory of your device.

58.1 Stack.clear

Syntax: Stack.clear <ptr_nexp>

The stack designated by <ptr_nexp> will be cleared.

58.2 Stack.create

Syntax: Stack.create N|S, <ptr_nvar>

Creates a new stack of the designated type (N=Number, S=String). The stack pointer is in <ptr_nvar>.

58.3 Stack.isEmpty

Syntax: Stack.isEmpty <ptr_nexp>, <nvar>

If the stack designated by <ptr_nexp> is empty the value returned in <nvar> will be 1. If the stack is not empty the value will be 0.

58.4 Stack.kill.last

Syntax: Stack.kill.last

Kills the last stack in the internal stacks list. Stacks are global. If you create a stack within a function, you can to kill it before leaving the function.

58.5 Stack.peek

Syntax: Stack.peek <ptr_nexp>, <nvar>|<svar>

Returns the top-of-stack value of the stack designated by <ptr_nexp> into the <nvar> or <svar>. The value will remain on the top of the stack.

The type of the value variable must match the type of the created stack.

58.6 Stack.pop

Syntax: Stack.pop <ptr_nexp>, <nvar>|<svar>

Pops the top-of-the-stack value designated by <ptr_nexp> and places it into the <nvar> or <svar>.

The type of the value variable must match the type of the created stack.

58.7 Stack.push

Syntax: Stack.push <ptr_nexp>, <nexp>|<sexp>

Pushes the <nexp> or <sexp> onto the top of the stack designated by <ptr_nexp>.

The type of value expression pushed must match the type of the created stack.

58.8 Stack.type

Syntax: Stack.type <ptr_nexp>, <svar>

The type (numeric or string) of the stack designated by <ptr_nexp> will be returned in <svar>. If the stack is numeric, the upper case character "N" will be returned. If the stack is a string stack, the upper case character "S" will be returned.

59 String Functions That Return a String

59.1 Bin$

Syntax: Bin$(<nexp>)

Returns a string representing the binary representation of the numeric expression.

59.2 Chr$

Syntax: Chr$(<nexp>, ...)

Return the character string represented by the values of list of numerical expressions. Each <nexp> is converted to a character. The expressions may have values greater than 255 and thus can be used to generate Unicode characters.

```
Print Chr$(16*4 + 3)    % Hexadecimal 43 is the character "C".       Prints: C
Print Chr$(945, 946)    % Decimal for the characters alpha and beta. Prints: αβ
Print Chr$(128194)      % 32-bit character open folder.              Prints: 📂
Print Chr$(55357,56514) % also creates a 32-bit char. by two 16-bit characters.
                        %                                            Prints: 📂
```

Android uses 16-bit characters so you should note, that a 32-bit character counts two 16-bit characters:

```
Len("📂") % Returns 2
```

See also:

https://www.w3schools.com/charsets/ref_html_utf8.asp (To choose!)

https://www.fileformat.info/info/unicode/char/search.htm (To get the 2 decimal values!)

59.3 Decode$

Syntax: Decode$(<type_sexp>, {<qualifier_sexp>}, <source_sexp>)

or

Syntax: Decode$(<charset_sexp>, <buffer_sexp>)

The first form of this command returns the result of decoding a string <source_sexp> that was encoded with Encode$(). The <type_sexp> and <qualifier_sexp> describe how the string was encoded. You must use the same type and qualifier that were used to encode the source string, or you may get unpredictable results.

Type	Qualifier	Default	Result
"ENCRYPT"	password	"" (empty)	Decrypts the source string using the password parameter. The encryption algorithm is "PBEWithMD5AndDES". This usage of **Decode$()** works the same way as the **Decrypt** command.
"DECRYPT"	password	""	Same as type "ENCRYPT" (decrypts, does not encrypt).
"ENCRYPT"	password	""	Same as type "DECRYPT" except the input and output are buffer strings. This is useful for decrypting binary data.
"DECRYPT"	password	""	Same as type "ENCRYPT".
"URL"	charset	"UTF-8"	Decodes the source string assumed to be in the format required by the HTML specification found at: http://www.w3.org/TR/html4/interact/forms.html#h-17.13.4.1. You should omit the charset parameter.

"BASE64"	charset	"UTF-8"	Decodes the source string holding the Base64 representation of binary data. See RFCs 2045 and 3548.

The type is required, but see the two-parameter form of **Encode$()** and **Decode$()**.

The type IS NOT case-sensitive: "BASE64", "base64", and "Base64" are all the same.

The qualifier is optional, but its comma is required.

If you supply the qualifier, whether password or charset, it IS case-sensitive.

The source string is decoded to a byte stream according to the type. Then the byte stream is converted to a BASIC! string (UTF-16) according to the character encoding (the charset parameter), which describes how to interpret the byte stream. The charset is always UTF-8 for decryption, and defaults to UTF-8 for the other types. The most common usage of this function is to omit the charset.

If the source string was encoded from binary data (with "ENCRYPT_RAW" or "BASE64"), the resulting BASIC! string will be a buffer string. When a string is used as a buffer, one byte of data is written into the lower 8 bits of each 16-bit character, and the upper 8 bits are 0. You can extract the binary data from the string, one byte at a time, using the **Ascii()** or **Ucode()** functions.

If the source string cannot be decoded (or decrypted) with the specified charset (or password), the function returns an empty string (""). You can call the **GetError()** function to get an error message.

See the two-parameter form of **Encode$()**, for a partial list of valid charsets.

Syntax: Decode$(<charset_sexp>, <buffer_sexp>)

The second form of this command decodes the buffer string <buffer_sexp> that was encoded using the <charset_sexp> and returns the result in a standard BASIC! string. A buffer string is a special use of the BASIC! string in which each 16-bit character consists of one byte of 0 and one byte of data.

If the source string cannot be decoded with the specified charset, the function returns an empty string (""). You can call the **GetError$()** function to get an error message.

If you attempt to **Decode$()** a string that is not a buffer string, you may get unexpected results. Besides the function **Encode$()**, the commands **Byte.read.buffer** and **BT.read.bytes** can write buffer strings. Your program can also build such strings directly, character-by-character.

If you read data from a file with **Byte.read.buffer**, you can use **Decode$()** to reassemble the bytes into BASIC! (UTF-16) strings. The charset specifies how the original string was encoded when it was written as bytes to the file.

For example, a binary file may have embedded text strings for names or titles. In order to allow Unicode, the text may be encoded. Let's say you read 32 bytes of binary data, consisting of 8 bytes of binary and 24 bytes of UTF-8-encoded text:

```
Byte.read.buffer file, 32, bfr$
namebfr$ = Mid$(bfr$, 9)
name$ = Decode$("UTF-8", namebfr$)
```

For encryption and URL- or Base64-decoding, see the three-parameter form of **Decode$()**.

59.4 Encode$

Syntax: Encode$(<type_sexp>, {<qualifier_sexp>}, <source_sexp>)

or

Syntax: Encode$(<charset_sexp>, <source_sexp>)

The first form of this command returns the string <source_sexp> encoded in one of several ways, as specified by the <type_sexp>. The <qualifier_sexp> usage depends on the type:

Type	Qualifier	Default	Result
"ENCRYPT"	password	"" (empty)	Encrypts the source string using the password parameter. The encryption algorithm is "PBEWithMD5AndDES". This usage of **Encode$()** works the same way as the **Encrypt** command. Use the **Decode$()** function to decrypt.
"DECRYPT"	password	""	Same as type "ENCRYPT" (encrypts, does not decrypt).
"ENCRYPT_RAW"	password	""	Same as type "ENCRYPT" except the input and output are buffer strings. This is useful for encrypting binary data.
"DECRYPT_RAW"	password	""	Same as "ENCRYPT_RAW" (encrypts, does not decrypt).
"URL"	charset	"UTF-8"	Encodes the source string using the format required by the HTML specification found at: http://www.w3.org/TR/html4/interact/forms.html#h-17.13.4.1. You should omit the charset parameter.
"BASE64"	charset	"UTF-8"	Encodes the source string into the Base64 representation of binary data. See RFCs 2045 and 3548. The simplest way to use this function is to omit the charset parameter.

The type is required, but see the two-parameter form of **Encode$()** and **Decode$()**.

The type IS NOT case-sensitive: "BASE64", "base64", and "Base64" are all the same.

The qualifier is optional, but its comma is required.

If you supply the qualifier, whether password or charset, it IS case-sensitive.

If the source string cannot be encoded (or encrypted) with the specified charset (or password), the function returns an empty string (""). You can call the **GetError$()** function to get an error message.

"ENCRYPT", "DECRYPT", and "URL" can be used on any BASIC! string. The string is converted to a byte stream, then the byte stream is encrypted or URL-encoded. The encrypted byte stream is converted to string format using the Base64 representation of binary data (see comment for "BASE64" in the table above. URL-encoded strings do not require this extra step.

"ENCRYPT_RAW" and "DECRYPT_RAW" are intended for use with binary data in a buffer string. A buffer string is a special use of the BASIC! string in which each 16-bit character consists of one byte of 0 and one byte of data. "ENCRYPT_RAW" can encrypt a buffer string or an ASCII text string, but using it on Unicode text will corrupt the string. After encryption, the result is returned in another buffer string.

"BASE64" also converts its string to a byte-stream before encoding it to Base64. The default conversion, using UTF-8, works with any BASIC! string; specifying another character set encoding may corrupt data.

Normally, you would use "BASE64" on binary data in a buffer string. In this case, you may specify any valid charset with no data corruption. The encoding string will change, but it can always be decoded using the same charset.

See the two-parameter form of **Encode$()**, for a partial list of valid charsets.

Syntax: Encode$(<charset_sexp>, <source_sexp>)

The second form of this command encodes the string <source_sexp> using the character encoding of the <charset_sexp> and returns the result in a buffer string. When a string is used as a buffer, one byte of data is written into the lower 8 bits of each 16-bit character, and the upper 8 bits are 0.

The charset specifies the rules used to convert the source string into a byte stream. The stream is written to a buffer string, one byte per character. The bytes are not reassembled into 16-bit characters.

The charsets "UTF-8", "UTF-16", "UTF-16BE", "UTF-16LE", "US-ASCII", and "ISO-8859-1" are always available. Your device may have additional charsets. The charset names are case-sensitive, but the standard charsets have aliases for convenience. For example, "utf8" is valid.

If the source string cannot be encoded with the specified charset, the function returns an empty string (""). You can call the **GetError$()** function to get an error message.

If you create a buffer string with **Encode$()**, you can write the bytes to a file with **Byte.write.buffer**.

For encryption and URL- or Base64-encoding, see the three-parameter form of **Encode$()**.

59.5 Format$

Syntax: Format$(<pattern_sexp>, <nexp>/<sexp>)

Returns a string with <nexp> or <sexp> formatted by the pattern <pattern_sexp>.

Keep in mind, that this function **does not round** the given number. BigDecimal numbers are stored as a String and can used directly.

Leading Sign	A negative (-) character for numbers < 0 or a space for numbers >= 0. The Sign and the Floating Character together form the **Floating Field**.
Floating Character	If the first character of the pattern is not "#" or "." or "-" then that character becomes a "floating" character. This pattern character is typically a "$". If no floating character is provided then a space character is used. See also **Overflow**, below.
Decimal Point	The pattern may have one optional decimal point character ("."). If the pattern has no decimal point, then only the whole number is output. Any digits that would otherwise appear after the decimal point are not output.
# Character (before decimal, or no decimal)	Each "#" is replaced by a digit from the number. If there are more "#" characters than digits, then the leading "#" character(s) are replaced by **space**(s).
# Character (after decimal point)	Each "#" is replaced by a digit from the number. If there are more "#" characters than significant digits, then the trailing "#" character(s) are replaced by **zero**(s). The number of "#" characters after the pattern decimal point specifies the number of decimal digits that will be output.
% Character (before decimal, or no decimal)	Each "%" is replaced by a digit from the number. If there are more "%" characters than digits, then the leading "%" character(s) are replaced by **zero**(s).
% Character (after decimal)	The "%" character is not allowed after the decimal point. This is a syntax error.
Non-pattern Characters	If any pattern character (other than the first) is not # or %, then that character is copied directly into the output. If the character would appear before the first digit of

	the number, it is replaced by a space. This feature is usually used for commas.
Overflow	If the number of digits exceeds the number of # and % characters, then the output has the ** characters inserted in place of the Floating Field.
Output Size	The number of characters output is always the number of characters in the pattern plus one for the sign, plus one more for the space if the pattern has no Floating Character.

The sign and the floating character together form a **Floating Field** two characters wide that always appears just before the first digit of the formatted output. If there are any leading spaces in the formatted output, they are placed before the floating field.

The "#" character generates leading spaces, not leading zeros. "##.###" formats 0.123 as ".123". If you want a leading zero, use a "%". For example "%.###", "#%.###", or "##%" all assure a leading zero.

Be careful mixing # and % characters. Doing so except as just shown can produce unexpected results.

The number of characters output is always the number of characters in the pattern plus the two floating characters.

Examples:

Function Call	Output	Width
Format$("##,###,###", 1234567)	1,234,567	12 characters
Format$("%%,%%%,%%%.#", 1234567.89)	01,234,567.8	14 characters
Format$("$###,###", 123456)	$123,456	9 characters
Format$("$###,###", -1234)	-$1,234	9 characters
Format$("$###,###", 12)	$12	9 characters
Format$("$%%%,%%%", -12)	-$000,012	9 characters
Format$("##.#", 0)	.0	6 characters
Format$("#%.#", 0)	0.0	6 characters
Format$("$###.##", -1234.5)	**234.50	8 characters

Example:

```
t = -15.97
mFormatString$ = "%#.#"
where = Is_In(".", mFormatString$)
mFrac$ = Mid$(mFormatString$, where + 1)
result$ = Format$( mFormatString$, Round(t, Len(mFrac$)))
PRINT result$
```

59.6 Format_using$

Syntax: Format_using$(<locale_sexp>, <format_sexp> { , <exp>}...)

Alias for **Using$()**. You can use the two equivalent functions to make your code easier to read. For example:

```
string$ = Format_using$("", "pi is not %d", Int(Pi()))
Print Using$("en_US", "Balance: $%8.2f", balance)
```

59.7 Hex$

Syntax: Hex$(<nexp>|<color_sexp>)

Returns a string representing the hexadecimal representation of the numeric expression <nexp>.

If the alternative <color_sexp> is specified, the function returns the color by the (#) hex color notation such as (#)ff00ff00 (_Lime). The source specified by <color_sexp> can be one of these options:

```
{Alpha,}Red,Green,Blue  % All these members in a range from 0 to 255
(comma delimited string)
or
_{Alpha,}ColorName
({comma delim.} string)
or
#{hn}hnhnhn
(hex. String)
```

Example:

```
hColor$ = "#" + Hex$("0,0,255") % (_Blue) returns "#ff0000ff"
```

59.8 Int$

Syntax: Int$(<nexp>)

Returns a string representing the integer part of the numeric expression.

59.9 Left$

Syntax: Left$(<sexp>, <count_nexp>)

Return the left-most characters of the string <sexp>. The number of characters to return is set by the count parameter, <count_nexp>.

- If the count is greater than 0, return <count_nexp> characters, counting from the left.
- If the count is less than 0, return all but <count_nexp> characters. The number to return is the string length reduced by <count_nexp>: **Left$(a$, -2)** is the same as **Left$(a$, Len(a$) - 2)**.
- If the count is 0, return an empty string ("").
- If the count is greater than the length of the string, return the entire string.

59.10 Lower$

Syntax: Lower$(<sexp>)

Returns <sexp> in all lower case characters.

59.11 Ltrim$

Syntax: Ltrim$(<sexp>{, <test_sexp>})

Exactly like **Trim$()**, except that **Ltrim$()** trims only the left end of the source string <sexp>.

59.12 Mid$

Syntax: Mid$(<sexp>, <start_nexp>{, <count_nexp>})

Return a substring of the string <sexp>, beginning or ending at the start position <start_nexp>. The first character of the string is at position 1. If the start position is 0 or negative, it is set to 1.

The count parameter is optional. If it is omitted, return all of the characters from the start position to the end of the string.

```
a$ = Mid$("dinner", 2)          % a$ is "inner"
```

Otherwise, the absolute value of the count specifies the length of the returned substring:

- If the count is greater than 0, begin at <start_nexp> and count characters to the **right**.
  That is, return the substring that **begins** at the start position.
  If the start position is greater than the length of the string, return an empty string ("").
- If the count is less than 0, begin at <start_nexp> and count characters to the **left**.
  That is, return the substring that **ends** at the start position.
  If the start position is greater than the length of the string, it is set to the end of the string.
- If the count is 0, return an empty string ("").

```
a$ = Mid$("dinner", 2, 3)       % a$ is "inn"
a$ = Mid$("dinner", 4, -3)      % a$ is "inn"
a$ = Mid$("dinner", 3, 0)       % a$ is ""
```

59.13 Ntrim$

Syntax: Ntrim$(<nexp>)

Returns a number as a string, which is trimmed to the smallest possible string length.

59.14 Oct$

Syntax: Oct$(<nexp>)

Returns a string representing the octal representation of the numeric expression.

59.15 Onex$

Syntax: Onex$(<locale_sexp>,<nexp>)

Returns the Ordinal Number Extension of a given numeric value. By default a point "." will be returned. If <locale_sexp> is "uk", "en" or "us", for values > 0 it returns "st", "nd", "rn" or "th".

For "se" (Swedish), returns ":a",":b" or ":e".

For "ir" (Irish), returns "ú".

For "nl", returns "e".

For "lat" (Latin), "gl" (Galician), "it" (Italian), "pt" (Portuguese) or "sp" (Spanish), returns only "º".

See also USING$ for Locale expressions.

Examples:

```
Onex$("uk", 0) returns "."
Onex$("", 1)       % returns ".". If your default language is maybe German
Onex$("se", 2)     % returns ":b"
Onex$("uk", 3)     % returns "rd"
Onex$("us", 11)    % returns "th"
Onex$("", 13)      % returns ".". If your default language is maybe French
Onex$("uk", 112)   % returns "th"
Onex$("us", 1003)  % returns "nd"
```

59.16 Replace$

Syntax: Replace$(<sexp>, <find_sexp>, <replace_sexp>){, <mode_sexp>}

Returns <sexp> with all instances of <find_sexp> replaced with <replace_sexp>.

Values for the optional <mode_sexp> are:

Value	Description
"_Default"	Replace **without** regular expressions as described above (default).
"_RegEx_All"	Replace **with** regular expressions **all** instances of <find_sexp>.
"_RegEx_First"	Replace **with** regular expressions **first** instance of <find_sexp>.
"_RegEx_All_IgnoreCase"	Replace **with** regular expressions **all** instances of <find_sexp>, but ignore case.
"_RegEx_First_IgnoreCase"	Replace **with** regular expressions **first** instance of <first_sexp>, but ignore case

Examples:

```
Result$ = Replace$("abcdefghijklmnabc", "abc", "rst", "_RegEx_All")
!-- returns "rstdefghijklmnrst"

Result$ = Replace$("abc:defghi,jklmn;abc", "[,;:]", "|", "_RegEx_All")
!-- returns "abc|defghi|jklmn|abc"

Result$ = Replace$("abcdefghijklmnabc", "abc", "rst", "_RegEx_First")
!-- returns "rstdefghijklmnabc"
```

59.17 Reverse$

Syntax: Reverse$(<sexp>)

Returns a copy of <sexp> with the order of the characters reversed.

Example:

```
Print Reverse$("Was it a cat I saw?") % Returns "?was I tac a ti saW"
```

59.18 Right$

Syntax: Right$(<sexp>, <count_nexp>)

Return the right-most characters of the string <sexp>. The number of characters to return is set by the count parameter, <count_nexp>.

- If the count is greater than 0, return <count_nexp> characters, counting from the right.
- If the count is less than 0, return all but <count_nexp> characters. The number to return is the string length reduced by <count_nexp>: **Right$(a$, -2)** is the same as **Right$(a$, Len(a$) - 2)**.
- If the count is 0, return an empty string ("").
- If the count is greater than the length of the string, return the entire string.

59.19 Rtrim$

Syntax: Rtrim$(<sexp>{, <test_sexp>})

Exactly like **Trim$()**, except that **Rtrim$()** trims only the right end of the source string <sexp>.

59.20 Spc$

Syntax: Spc$(<nexp>{, <sexp>})

Returns a string with <nexp> spaces. If <sexp> is set, the function returns <nexp> times the <sexp> string. For example, Chr$(9) can insert tabs instead of spaces.

59.21 Str$

Syntax: Str$(<nexp>)

Returns the string representation of <nexp>.

59.22 Trim$

Syntax: Trim$(<sexp>{, <test_sexp>})

Returns <sexp> with leading and trailing occurrences of <test_sexp> removed.

The expression to trim off, <test_sexp>, is optional. If omitted, all leading and trailing whitespace is removed. That is, the default <test_sexp> is the regular expression "\s+", which means "all whitespace". To use this regular expression in a BASIC! string, you must write it "\\s+" (escape the backslash).

As with the **Word$()** function and the **Split** command, the <test_exp> is a regular expression. See **Split** for a note about Regular Expressions.

59.23 Upper$

Syntax: Upper$(<sexp>)

Returns <nexp> in all upper case characters.

59.24 Using$

Syntax: Using$({<locale_sexp>} , <format_sexp> { , <exp>}...)

Returns a string, using the locale and format expressions to format the expression list.

This function gives BASIC! programs access to the Formatter class of the Android platform. You can find full documentation at http://developer.android.com/reference/java/util/Formatter.html.

The <locale_sexp> is a string that tells the formatter to use the formatting conventions of a specific language and region or country. For example, "en_US" specifies American English conventions.

The <format_sexp> is a string that contains *format specifiers*, like "%d" or "%7.2f," that tell the formatter what to do with the expressions that follow.

The format string is followed by a list of zero or more expressions. Most format specifiers take one argument from the list, in order. If you don't provide as many arguments as your format string needs, you will get a detailed Java error message.

Each expression must also match the type of the corresponding format specifier. If you try to apply a string format specifier, like "%-7s", to a number, or a floating point specifier, like "%5.2f" to a string, you will get a Java error message.

59.24.1 Locale expression

The **Using$()** function can localize the output string based on language and region. The locale specifies the language and region with standardized codes. The <locale_sexp> is a string containing zero or more codes separated by underscores.

The function accepts up to three codes. The first must be a language code, such as "en", "de", or "ja". The second must be a region or country code, such as "FR", "US", or "IN". Some language and country combinations can accept a third code, called the "variant code".

The function also accepts the standard three-letter codes and numeric codes for country or region. For example, "fr_FR", "fr_FRA", and "fr_250" are all equivalent.

If you want to use the default locale of your Android device, make the <locale_exp> an empty string (""), or leave it out altogether. If you leave it out, you must keep the comma: **Using$(, "%f", x)**

If you make a mistake in the <locale_sexp>, you may get an obscure Java error message, but more likely your locale will be ignored, and your string will be formatted using the default locale of your device.

Android devices do not support all possible locales. If you specify a valid locale that your device does not understand, your string will be formatted using the default locale.

59.24.2 Format expression

If you are familiar with the *printf* functions of C/C++, Perl, or other languages, you will recognize most of format specifiers of this function. The format expression is exactly the same as format string of the Java *Formatter* class, or the *format(String, Object…)* method of the Java *String*, with two exceptions: Boolean format specifiers are not supported, and hex hash specifiers are limited to numeric and string types.

If you have not programmed in one of those other languages, this will be your introduction to a powerful tool for formatting text.

A format expression is a string with embedded format specifiers. Anything that is not a format specifier is copied literally to the function output string. Each embedded format specifier is replaced with the value of an expression from the list, formatted according to the specifier. For example:

```
Print Using$("","Pi is approximately %f.", PI())  % function call
Pi is approximately 3.141593.                     % printed output for
                                                  % English locale
```

The <locale_exp> is "", meaning "use my default locale".
The <format_exp>, "Pi is approximately %f", has one format specifier, "%f".
"%f" means, "use the default decimal floating point output format".
The expression list has one item, the math function **Pi()**.
In the output, "%f" is replaced by the value of the the **Pi()** function.
Your output may be different if your locale language is not English.

59.24.2.1 Format Specifiers

Here is a brief summary of the available format specifiers:

For this type of data	Use these formats	Comments
String	%s %S	%S forces output to upper-case
Number	%f %e %E %g %G %a %A	Standard BASIC! numbers are floating point Use %f for decimal output: "1234.567" Use %e or %E for exponential notation: "1.234e+03" %E writes upper-case: "1.234E+03" %g (%G) lets the system choose %f or %e (%E) %a and %A are "hexadecimal floating point"
Integer	%d %o %x %X	USING$ can use some math functions as integers Use %d for decimal, %o for octal, %x %X for hex %x writes lower-case abcdef, %X writes upper-case
Special integer	%c %C %t	These specifiers can operate on an integer %c %C output a character, %C writes upper-case

For this type of data	Use these formats	Comments
		%t represents a family of time format specifiers
None	%% %n	These specifiers do not read the expression list %% writes a single "%" to the output %n writes a newline, exactly the same as \n

For more information about %a and %A, see the Android documentation linked above.
Android's %b and %B are not supported because BASIC! has no Boolean type.
Android's %h and %H hash code specifiers are limited to strings and numbers in BASIC!.
For an explanation of **Using$()** with integer format specifiers, see below.

There is a whole family of time format specifiers: **%t<x>** where **<x>** is another letter. They operate on an integer, which they interpret as the number of milliseconds since the beginning of January 1, 1970, UTC (the "epoch"). You can apply time format specifiers to the output of the **Time()** functions. Note, however, that the **%t** time specifiers use your local timezone, not the **TimeZone.set** value.

There are more than 30 time format specifiers. A few examples appear below, but to get the full list you should read the Android documentation linked above.

```
Print Using$("", "The time is: %tI:%<tM:%<tS %<Tp", Time()) % the hard way
Print Using$("", "The time is: %tr", time())               % same thing!
02:27:16 PM                                               % sample of printed output

t = Time(2001, 2, 3, 4, 5, 6)               % set 2001/02/03 04:05:06, local timezone
Print Using$("sv", "%tA", Int(t))           % day in Swedish, prints "lördag"
Print Using$("es", "%tB", Int(t))           % month in Spanish, prints "febrero"
Print Using$("", "%tY/%tm/%td", Int(t) , Int(t), Int(t))    % prints
                                                            % "2001/02/03"
Print Using$( , "%tY/%<tm/%<td", Int(t))                    % prints "2001/02/03"
Print Using$("", "%tF", t)                                  % prints "2001:02:03"
Print Using$("en_GB", "%tH:%<tM:%<tS", Int(t))              % prints "04:05:06"
Print Using$("in_IN", "%tT", Int(t))                        % prints "04:05:06"
```

Note: Date and time are printed for your local timezone, regardless of either the **TimeZone.set** setting or the locale parameter. Try the same set of examples with **TimeZone.set "UTC"**. Unless that is your local timezone, a different hour and perhaps even a different day will be displayed.

59.24.2.2 Optional Modifiers

The format specifiers can be used exactly as shown in the table. They have default settings that control how they behave. You can control the settings yourself, fine-tuning the behavior to suit your needs.

You can modify the format specifiers with *index*, *flags*, *width*, and *precision*, as shown in this example:

"%3$-,15.4f"							
"%	3$	-,	15	.	4	f	"
	<index>	<flags>	<width>		<precision>	<specifier>	

59.24.2.3 Index

Normally the format specifiers are applied to the arguments in order. You can change the order with an argument index. An index a number followed by a **$** character. The argument index **3$** specifies the third argument in the list.

```
Print Using$("", "%3$s %s %s", "a", "b", "c")     % prints "c a b"
```

The special argument index "<" lets you reuse an argument.

```
Print Using$("", "%o %<d %<h", Int(64) )          % prints "100 64 40"
```

In the last example, there is only one argument, but three format specifiers. This is not an error because the argument is reused.

59.24.2.4 Flags

There are six flags:

-	left-justify; if no flag, right-justify
+	always show sign; if no flag, show "-" but do not show "+"
0	pad numbers with leading zeros; if no flag pad with spaces
,	use grouping separators for large numbers
(	put parentheses around negative values
#	alternate notation (leading 0 for octal, leading 0x for hexadecimal)

59.24.2.5 Width

The **width** control sets the minimum number of characters to print. If the value to format is longer than the width setting, the entire value is printed (unless it would exceed the **precision** setting). If the value to format is shorter than the **width** setting, it is padded to fill the width. By default, it is padded with spaces on the left, but you change this with the "-" and "0" flags.

59.24.2.6 Precision

The **precision** control means different things to different data types.

For floating point numbers, **precision** specifies the number of characters to print after the decimal point, padding with trailing zeros if needed.

For string values, it specifies the maximum number of characters to print. If **precision** is less than **width**, only **precision** characters are printed.

```
"%4s",   "foo" prints " foo"
"%-4s",  "foo" prints "foo "
"%4.2s", "foo" prints "fo"
```

The **precision** control is not valid for other types.

In the example above, **%-,15.4f**:

The **flags** "-" and "," mean "left-justify the output" and "use a thousands separator".
The **width** is 15, meaning the output is to be at least 15 characters wide.
The **precision** is 4, so there will be exactly four digits after the decimal point.

The whole format specifier means, "format a floating point number (%f) left-justified ("-") in a space 15 characters wide, with 4 characters after the decimal point, with a thousands separator (",")".

The characters used for the decimal point and the thousands separator depend on the locale:

```
"1,234.5678     " for locale "en"
"1 234,5678     " for locale "fr"
"1.234,5678     " for locale "it"
```

59.24.3 Integer values

BASIC! has only double-precision floating point numbers. It does not have an integer type. The **Using$()** function supports format specifiers ("%d", "%t", "%x") that apply only to integer values.

The functions that can produce integer values for **Using$()** are:

```
Int()    Bin()    Oct()    Hex()    Ceil()
Floor()  Ascii()  Ucode()  Shift()  Time()
Band()   Bor()    Bxor()
```

59.25 Word$ / Word_All$

Syntax: Word$(<source_sexp>, <n_nexp> {, <test_sexp>})

or

Syntax: Word_all$(<source_sexp>, <n_nexp> {, <test_sexp>})

This function returns a word from a string. The <source_sexp> string is split into substrings at each location where <test_sexp> occurs. The <n_nexp> parameter specifies which substring to return; numbering starts at 1. The <test_sexp> is removed from the result. The <test_sexp> parameter is an optional Regular Expression; if it is not given, the source string is split on whitespace. Specifically, the default <test_sexp> is "\s+". Whitespace is not only the Space character. It can also be one of 25 Unicode characters. CR (carriage return) and LF (line feed) are also whitespace. See also https://en.wikipedia.org/wiki/Whitespace_character.

Thus, **Word$** and **Word_all$** have to test against for many characters which slows down the speed of these command. Use " " as <test_sexp> if you want to split by a simple Space character.

Leading and trailing occurrences of <test_sexp> are stripped from <source_sexp> before it is split. If <n_nexp> is less than 1 or greater than the number of substrings found in <source_sexp>, then an empty string ("") is returned. Two adjacent occurrences of <test_sexp> in <source_sexp> result in an empty string; <n_nexp> may select this empty string as the return value.

Examples:

```
string$ = "The quick brown fox"
result$ = Word$(string$, 2)                  % result$ is "quick"

string$ = ":a:b:c:d"
delimiter$ = ":"
Split array$[], string$, delimiter$          % array$[1] is ""
result$ = Word$(string$, 1, delimiter$)      % result$ is "a", not ""
```

This function is similar to the **Split** command.

Word_all$** is the same as **Word$ except that it will not trim leading and trailing empty words.

60 String Functions That Return a Number

60.1 Ascii

Syntax: Ascii(<sexp>{, <index_nexp>})

Returns the ASCII value of one character of <sexp>. By default, it is the value of the first character. You can use the optional <index_nexp> to select any character. The index of the first character is 1.

A valid ASCII value is between 0 and 255. If <sexp> is an empty string ("") the value returned will be 256 (one more than the largest 8-bit ASCII value). For non-ASCII Unicode characters, **Ascii()** returns invalid values; use **Ucode()** instead.

60.2 Bin

Syntax: Bin(<sexp>)

Returns the numerical value of the string expression <sexp> interpreted as a binary integer. The characters of the string can be only binary digits (0 or 1), with an optional leading sign ("+" or "-"), or the function generates a runtime error.

60.3 Ends_with

Syntax: Ends_with(<sub_sexp>, <base_sexp>)

Determines if the substring <sub_sexp> exactly matches the end of the base string <base_sexp>.

If the base string ends with the substring, the function returns the index into the base string where the substring starts. The value will always be >= 1. If no match is found, the function returns 0.

60.4 Hex

Syntax: Hex(<sexp>)

Returns the numerical value of the string expression <sexp> interpreted as a hexadecimal integer. The characters of the string can be only hexadecimal digits (0-9, a-h, or A-H), with an optional leading sign ("+" or "-"), or the function generates a runtime error.

60.5 Is_in

Syntax: Is_in(<sub_sexp>, <base_sexp>{, <start_nexp>})

Returns the position of an occurrence of the substring <sub_sexp> in the base string <base_sexp>.

If the optional start parameter <start_nexp> is not present then the function starts at the first character and searches forward.

If the start parameter is >= 0, then it is the starting position of a forward (left-to-right) search. The left-most character is position 1. If the parameter is negative, it is the starting position of a reverse (right-to-left) search. The right-most character is position -1.

If the substring is not in the base string, the function returns 0. It can not return a value larger than the length of the base string.

60.6 Is_number

Syntax: Is_number(<sexp>)

Tests a string expression <sexp> in the same way as **Val()** and returns a logical value:

- TRUE (non-zero) if **Val()** would successfully convert the string to a number
- FALSE (0) if **Val()** would generate a run-time error. For example, **Val("name")** generates a run-time error, but **Is_number("name")** returns FALSE.

If **Val()** would report a syntax error, **Is_number()** reports a syntax error. For example, **Is_number()**, **Is_number(num)**, and **Is_number(5)** are syntax errors.

60.7 Len

Syntax: Len(<sexp>)

Returns the length of the <sexp>.

60.8 Oct

Syntax: Oct(<sexp>)

Returns the numerical value of the string expression <sexp> interpreted as an octal integer. The characters of the string can be only octal digits (0-7), with an optional leading sign ("+" or "-"), or the function generates a runtime error.

60.9 Starts_with

Syntax: Starts_with(<sub_sexp>, <base_sexp>{, <start_nexp>})

Determines if the substring <sub_sexp> exactly matches the part of the base string <base_sexp> that starts at the position <start_nexp>. The <start_nexp> parameter is optional; if it is not present then the default starting position is 1, the first character, so the base string must start with the substring. If present, <start_nexp> must be >= 1.

The function returns the length of the matching substring. If no match is found, the function returns 0.

60.10 Ucode

Syntax: Ucode(<sexp>{, <index_nexp>})

Returns the Unicode value of one character of <sexp>. By default, it is the value of the first character. You can use the optional <index_nexp> to select any character. The index of the first character is 1.

If <sexp> is an empty string (""), the value returned will be 1000000.0. If the selected character of <sexp> is a valid ASCII character, this function returns the same value as **Ascii()**.

In case of 32-bit characters, `Ucode("🗁", 1)` returns 55357 and `Ucode("🗁", 2)` returns 56514.

60.11 Ucode32

Syntax: Ucode32(<sexp>)

Returns the Unicode value of the first 16- or 32-bit character of <sexp>.

If <sexp> is an empty string (""), the value returned will be 1000000.0. If the selected character of <sexp> is a valid ASCII character, this function returns the same value as **ASCII()**.

In case of 32-bit characters `Ucode32("🗁")` returns 128194.

60.12 Val

Syntax: Val(<sexp>)

Returns the numerical value of the string expression <sexp> interpreted as a signed decimal number. If the string is empty ("") or does not represent a number, the function generates a runtime error.

- A sign ("+" or "-"), a decimal point ("." only), and an exponent (power of 10) are optional.
- An exponent is "e" or "E" followed by a number. The number may have a sign but no decimal point.
- The string may have leading and/or trailing spaces, but no spaces between any other characters.

61 String Commands

See also the various String Functions.

61.1 Decrypt

Syntax: Decrypt <pw_sexp>, <encrypted_sexp>, <decrypted_svar>

Decrypts the encrypted string <encrypted_sexp> using the password <pw_sexp>. The result is placed in <decrypted_svar>. The encryption algorithm used is "PBEWithMD5AndDES".

The password parameter is optional, but its comma is required. Omitting the password is the same as using an empty string, "".

If the source string cannot be decoded with the specified password, the result is an empty string (""). You can call the **GetError$()** function to get an error message.

This command is the same as **Decode$("ENCRYPT", <pw_sexp>, <source_sexp>)**.

61.2 Encrypt

Syntax: Encrypt {<pw_sexp>}, <source_sexp>, <encrypted_svar>

Encrypts the string <source_sexp> using the password <pw_sexp>. The result is placed into the variable <encrypted_svar>. The encryption algorithm used is "PBEWithMD5AndDES".

The password parameter is optional, but its comma is required. Omitting the password is the same as using an empty string, "".

If the source string cannot be encrypted, the result is an empty string (""). You can call the **GetError$()** function to get an error message.

This command is the same as **Encode$("ENCRYPT", <pw_sexp>, <source_sexp>)**.

61.3 Join / Join.all

Syntax: Join <source_array$[]>, <result_svar> {, <separator_sexp>{, <wrapper_sexp>}}

or

Syntax: Join <source_array[]>, <result_svar> {, <separator_sexp>{, <wrapper_sexp>}}

or

Syntax: Join.all <source_array$[]>, <result_svar> {, <separator_sexp>{, <wrapper_sexp>}}

or

Syntax: Join.all <source_array[]>, <result_svar> {, <separator_sexp>{, <wrapper_sexp>}}

Split and **Join** are complementary operations. **Split** separates a string into parts and put the parts in an array. **Join** builds a string by combining the elements of an array.

The elements of <source_array$[]> or <source_array[]> are joined together as a single string in the <result_svar>. Using a numeric array, all numbers will be trimmed to the smallest possible string length. By default, the source elements are joined with nothing between them or around them.

You may specify optional modifiers that add characters to the string. Copies of the separator string

<separator_sexp> are written between source elements. Copies of the wrapper string <wrapper_sexp> are placed before and after the rest of the result string.

The **Join** command omits any empty source elements. The **Join.all** command includes all source elements in the result string, even if they are empty. There is no difference between the two commands unless you specify a non-empty separator string. **Join.all** places copies of the separator between all of the elements, including the empty ones.

An example of an operation that uses both separators and wrappers is a CSV string, for "comma-separated values".

```
InnerPlanets$ = "Mercury Venus Earth Mars"
Split IP$[], InnerPlanets$
Join  IP$[], PlanetsCSV$, "\",\"", "\""
Print PlanetsCSV$
```

This prints the string **"Mercury","Venus","Earth","Mars"** (including all of the quotes). The separator puts the `",` between planet names, and the wrapper puts the `"` at the beginning and end of the string.

61.4 Split / Split.all

Syntax: Split <result_array$[]>, <sexp> {, <test_sexp>}

or

Syntax: Split.all <result_array$[]>, <sexp> {, <test_sexp>}

Split and **Join** are complementary operations. **Split** separates a string into parts and put the parts in an array. **Join** builds a string by combining the elements of an array.

Splits the source string <sexp> into multiple strings and place them into <result_array$[]>. The array is specified without an index. If the array exists, it is overwritten. Otherwise a new array is created. The result is always a one-dimensional array.

The string is split at each location where <test_sexp> occurs. The <test_sexp> occurrences are removed from the result strings. The <text_sexp> parameter is optional; if it is not given, the string is split on whitespace. Omitting the parameter is equivalent to specifying "\\s+". Whitespace is not only the Space character. It can also be one of 25 Unicode characters. CR (carriage return) and LF (line feed) are also whitespace. See also https://en.wikipedia.org/wiki/Whitespace_character.

Thus, **Split** and **Split.all** have to test against for many characters which slows down the speed of these command. Use " " as <test_sexp> if you want to split by a simple Space character.

If the beginning of the source string matches the test string, the first element of the result array will be an empty string. This differs from the **Word$()** function, which strips leading and trailing occurrences of the test string from the source string before splitting.

Two adjacent occurrences of the test expression in the source expression result in an empty element somewhere in the result array. The **Split** command discards these empty strings if they occur at the end of the result array. To keep these trailing empty strings, use the **Split.all** command.

Example:

```
string$ = "a:b:c:d"
delimiter$ = ":"
Split result$[], string$, delimiter$

Array.length length, result$[]
For i = 1 To length
Print result$[i] + " ";
```

```
Next i
Print ""
```

Prints: `a b c d`

Note: The <test_sexp> is actually a Regular Expression. If you are not getting the results that you expect from the <test_sexp> then you should examine the rules for Regular Expressions at:

http://developer.android.com/reference/java/util/regex/Pattern.html

62 Superuser Commands

The Su commands are identical to the System commands except that for Su commands:

- The initial working directory is the root directory, */*.
- The resulting command shell has superuser privileges.

Some devices do not allow the **Su.Open** statement to execute successfully. In this case, the statement fails, issuing an error message such as the following:

```
SU Exception: Cannot run program "su": error=13, Permission denied
```

62.1 Su.close

Syntax: Su.close

Exits the Superuser mode.

62.2 Su.open

Syntax: Su.open

Requests Superuser permission. If granted, opens a shell to execute system commands. The working directory is set to /. If you open a command shell with either **Su.open** or **System.open**, you can't open another one of either type without first closing the open one.

62.3 Su.read.line

Syntax: Su.read.line <svar>

Places the next available response line into the string variable.

62.4 Su.read.ready

Syntax: Su.read.ready <nvar>

Tests for responses from a **Su.write** command. If the result is non-zero, then response lines are available. Not all Superuser commands return a response. If there is no response returned after a few seconds then it should be assumed that there will be no response.

62.5 Su.write

Syntax: Su.write <sexp>

Executes a Superuser command.

63 System Commands

BASIC! runs on devices that use Android, a Linux-based operating system. System commands allow BASIC! to send text commands directly to the operating system. These commands are essentially Linux commands.

The BASIC! statements in this section open a command shell. This is an environment with a saved context. The context remembers the results of a given command so that they affect subsequent commands. For example, if you change the working directory of a shell, the change remains in effect for subsequent statements.

The BASIC! program must open the command interface before using it, and should close it when operations are complete. The command interface works by sending string expressions and then waiting for responses to be read back, one line at a time, into string variables. The time this takes is device dependent. The shell does not respond to some commands at all, so the BASIC! program should loop and be ready to time-out in an orderly manner if the shell does not respond.

The System statements use a shell to the Android operating system. The SU statements use a shell with superuser privileges. Only one can be active at any time.

63.1 System.close

Syntax: System.close

Exits the System Shell mode. The shell's environment and context is discarded. Opening a new shell, with the **System** or **SU** statements, will not have any information from a previous shell, unless the program has saved it in, for example, a file or database.

63.2 System.open

Syntax: System.open

Opens a shell to execute system commands. The working directory is set to "**rfo-basic**". If the working directory does not exist, it is created. If you open a command shell with either **Su.open** or **System.open**, you can't open another one of either type without first closing the open one.

63.3 System.read.line

Syntax: System.read.line <svar>

Places the next available response line into the string variable. The returned string does not include a line terminator.

The BASIC! program should use **System.read.ready** statement to see if a response is available. If a response is not available, **System.read.line** sets <svar> to an empty string.

If **System.read.ready** indicates that the command shell has responded, then **System.read.line** can be called repeatedly, without further polling or pauses, to retrieve each line of the response in sequence. The BASIC! program should be written to take into account how many lines a command will return, or continue calling **System.read.line** until it returns an empty string or a string known to be the last line of the response. Some commands return blank lines; these also cause **System.read.line** to set <svar> to an empty string.

63.4 System.read.ready

Syntax: System.read.ready <nvar>

Tests for responses from a **System.write** command. If the result is non-zero, then response lines are available. Not all System commands return a response. If there is no response returned after a period of time (maybe a second) then it should be assumed that there will be no response.

63.5 System.write

Syntax: System.write <sexp>

Sends a System command <sexp> to the shell. The command in the string does not need to end with a line terminator.

This example will request the listing of a specified directory:

```
System.write "ls -al " + DirectoryName$
```

64 Text Commands

SMS commands will be only supported until Android version 10, because of Android security management.

64.1 Sms.send

Syntax: Sms.send <number_sexp>, <message_sexp>

The SMS message in the string expression <message_sexp> will be sent to number in the string expression <number_sexp>. This command does not provide any feedback about the sending of the message. The device must be connected to a cellular network to send an SMS message.

64.2 Sms.rcv.init

Syntax: Sms.rcv.init

Prepare to intercept received SMS using the **Sms.rcv.next** command.

64.3 Sms.rcv.next

Syntax: Sms.rcv.next <svar>

Read the next received SMS message from received SMS message queue in the string variable.

The returned string will contain "@" if there is no SMS message in the queue.

The **Sms.rcv.init** command must be called before the first **Sms.rcv.next** command is executed.

Example:

```
Sms.rcv.init
Do
 Do % Loop until SMS received
  Pause 5000 % Sleep of 5 seconds
  Sms.rcv.next m$ % Try to get a new message
 Until m$ <> "@" % "@" indicates no new message
 Print m$ % Print the new message
Until 0   % Loop forever
```

65 Time Functions

The **TimeZone** commands allow you to manage the timezone used by the **Time** command and the **Time(…)** function. They do not affect the no-parameter **Time()** function. They affect only your BASIC! program, not any other time-related operation on your device.

There are also BigDecimal commands such as **`BigD.nanoTime`**, **`BigD.time`** and **`BigD.date`**.

65.1 Clock

Syntax: Clock({nano_nexpr})

Returns the time in milliseconds since the last start, including deep sleep. This clock is guaranteed to be monotonic, and continues to tick even when the CPU is in power saving modes. Therefore, it is the recommend way for general purpose interval timing. If the optional <nano_nexpr> is greater than 0, nanoseconds are returned. The default is 0.

To compare two nano Time values:

```
tic1 = Clock(1)
...
tic2 = Clock(1)
if tic2 - tic1 < 0 ...
```

Use `tic2 - tic1 < 0` instead of `tic2 < tic1`, because of the possibility of numerical overflow.

Note that one day has 86,400,000,000,000 nanoseconds, which is a 14 digit number. The double values only have 15 significant digits. **`BigD.nanoTime`** is also an option, but it has a different time base.

65.2 Time()

Syntax: Time()

or

Syntax: Time(<year_exp>, <month_exp>, <day_exp>, <hour_exp>, <minute_exp>, <second_exp>)

The first form of this command returns the time in milliseconds since 12:00:00 AM, January 1, 1970, UTC (the "epoch"). The time interval is the same everywhere in the world, so the value is not affected by the **TimeZone** command.

The second form of this command uses parameters to specify a moment in time. The specification is not complete, as it does not include the timezone. You may specify a timezone with the **TimeZone** command. If you do not specify a timezone, your local timezone is used.

The parameter expressions may be either numeric expressions or string expressions. This is an unusual aspect as it isn't allowed anywhere else in BASIC!. If a parameter is a string, then it must evaluate to a number: digits only, one optional decimal point somewhere, optional leading sign, no embedded spaces. If the string parameter does not follow the rules, BASIC! reports a syntax error, like using a string in a place that expects a numeric expression.

Time(…) (the function) and **Time** (the command) are inverse operations. **Time(…)** can take the first six return parameters of the **Time** command directly as input parameters.

With the **Using$()** or **Format_using$()** functions, you can express a moment in time as a string in many different ways, formatted for your locale.

66 Time Commands

There are also BigDecimal commands such as **BigD.nanoTime**, **BigD.time** and **BigD.date**.

66.1 Time

Syntax: Time {<time_nexp>,} Year$, Month$, Day$, Hour$, Minute$, Second$, WeekDay, isDST

Returns the current (default) or specified date, time, weekday, and Daylight Saving Time flag in the variables.

You can use the optional first parameter (<time_nexp>) to specify what time to return in the variables. It is a numeric expression number of milliseconds from 12:00:00 AM, January 1, 1970, UTC, as returned by the **Time()** functions. It may be negative, indicating a time before that date.

The day/date and time are returned as two-digit numeric strings with a leading zero when needed, except Year$ which is four characters.

The WeekDay is a number from 1 to 7, where 1 means Sunday. You can use it to index an array of day names in your language of choice.

The isDST flag is
1 if the current or specified time is in Daylight Saving Time in the current timezone
0 if the time is not in Daylight Saving Time (is Standard Time)
-1 if the system can't tell if the time is in DST

The current timezone is your local timezone unless you change it with the **TimeZone** commands.

All of the return variables are optional. That is, you can omit any of them, but if you want to return only some of them, you need to retain their position by including commas for the omitted return variables. For example:

```
t = Time(2001, 2, 3, 4, 5, 6)
Time t, Y$, M$, D$        % sets only the year, month, and day
Time t, Y$, M$, D$,,,, W  % adds the day of the week (7, Saturday)
```

To do the same with the current time, leave out both the first parameter and its comma:

```
Time ,, day$,,,, wkday    % returns the today's day and weekday
```

66.2 TimeZone.get

Syntax: TimeZone.get <tz_svar>

Returns the current timezone in the string variable. This is the default timezone for your device and location, unless you have changed it with **TimeZone.set**.

66.3 TimeZone.list

Syntax: TimeZone.list <tz_list_pointer_nexp>

While time zones are defined by international standards, the only ones that matter to your program are those recognized by your device. This command returns all valid timezone strings, putting them in the list that <tz_list_pointer_nexp> points at. The previous contents of the list are discarded. If the pointer does not specify a valid string list, and the expression is a numeric variable, a new list is created and the variable is set to point to the new list.

66.4 TimeZone.set

Syntax: TimeZone.set { <tz_sexp> }

Sets the timezone for your program. If you don't specify a timezone, it is set to the default for your device, which is based on where you are. If you specify a timezone your device does not recognize, it is set to "GMT". (In Android, GMT is exactly the same as UTC).

67 Timer and Scheduler Commands

67.1 Timer

You can set a timer that will interrupt the execution of your program at some set time interval. When the timer expires, BASIC! jumps to the statements following the **OnTimer:** label. When you have done whatever you need to do to handle this Timer event, you use the **Timer.resume** command to resume execution of the program at the point where the timer interrupt occurred.

The timer cannot interrupt an executing command. When the timer expires, the current command is allowed to complete. Then the timer interrupt code after the **OnTimer:** label executes. If the current command takes a long time to finish, it may appear that your timer is late.

The timer cannot interrupt another interrupt. If the timer expires while any interrupt event handler is running, the **OnTimer:** interrupt will be delayed. If the timer expires while the **OnTimer:** interrupt handler is running, the timer event will be lost. The **OnTimer:** interrupt code must exit by running **Timer.resume**, or the timer interrupt can occur only once.

67.1.1 OnTimer:

Syntax: OnTimer:

Label for an interrupt handler for the timer timeout. When done, execute the **Timer.resume** command to resume the interrupted program.

67.1.2 Timer.clear

Syntax: Timer.clear

Clears the repeating timer. No further timer interrupts will occur.

67.1.3 Timer.resume

Syntax: Timer.resume

This statement should be placed at the end of the interrupt handler at **OnTimer:**.

67.1.4 Timer.set

Syntax: Timer.set <interval_nexp>

Sets a timer that will repeatedly interrupt program execution after the specified time interval. The interval time units are milliseconds. The program must also contain an **OnTimer:** label.

Example:

```
n = 0
Timer.set 2000

Do
Until n = 4
Timer.clear
Print "Timer cleared. No further interrupts."
Do
Until 0

ONTIMER:
n = n + 1
Print n * 2; " seconds"
Timer.resume
```

67.2 Scheduler

You can set a scheduler that will trigger an execution interrupt of your program at a set time or some interval. When the scheduler triggers, your program jumps to the statements following the **OnSched:** label. After your program has handled the scheduler event, the **Sched.resume** command will resume execution of the program at the point where the scheduler interrupt occurred.

The scheduler cannot interrupt an executing command. When the scheduler triggers, the current command is allowed to complete. Then the scheduler interrupt code after the **OnSched:** label executes. If the current command takes a long time to finish, it may appear that your scheduler is late.

The scheduler cannot interrupt another interrupt. If the scheduler triggers while any interrupt event handler is running, the **OnSched:** interrupt will be delayed. If the scheduler triggers while the **OnSched:** interrupt handler is running, the scheduler event will be lost. The **OnSched:** interrupt code must exit by running **Sched.resume**, or the scheduler interrupt will occur only once.

67.2.1 Sched.clear

Syntax: Sched.clear

Clears the scheduler. No further scheduler interrupts will occur.

67.2.2 Sched.resume

Syntax: Sched.resume

This statement should be placed at the end of the interrupt handler at **OnSched:**.

67.2.3 Sched.set

Syntax: Sched.set <firstInterrupt_nexp>, <interval_nexp> {,<date_flag_nexp>}

Sets a scheduler that will interrupt program execution after the <firstInterrupt_nexp> specified time or date in milliseconds. The parameter <interval_nexp> sets a repetitive program execution interrupt with the specified time interval in milliseconds. The optional flag <date_flag_nexp> defines whether <firstInterrupt_nexp> is a date driven time (>0) or a time interval (≤0). The date driven time is default. If <firstInterrupt_nexp> and <interval_nexp> are equal and <date_flag_nexp> is 0, the functionality is the same as **Timer.set**. That is a good option, if you need two timers. The program must also contain an **OnSched:** label.

Examples:

```
t = Time(2018, 1, 1, 0, 0, 1)
Sched.set t, 0              % Happy New Year 2018! One time interrupt. If your
                            % device is switched off at this date, the interrupt
                            % will occur at the next program start.

t = TIME() + 4000
Sched.set t, 2000, 1        % First Interrupt at current time + 4 seconds and
                            % the interrupt repeats every 2 seconds.

Sched.set 0, 2000, 0        % Interrupt immediately and repeats every 2 seconds.

Sched.set 2000, 2000, 0     % This is the same as TIMER.SET 2000

Sched.set 2000, 0, 0        % Interrupt one time in 2 seconds.
```

67.2.4 OnSched:

Syntax: OnSched:

Label for an interrupt handler which traps the scheduler events. When done, execute the **Sched.resume** statement to resume the interrupted program.

68 USB Commands

The Universal Serial Bus enables data transfer via serial communication.

Supported drivers:

- FTDI FT232
- Silicon Labs CP210x
- Prolific PL2303HX (at least HX version)
- CH340/CH341
- Generic CDC driver

The serial communication has several protocol variants.

Known issue: The interrupt queue can be overloaded so an interrupt may be jumped over.

If you run into trouble, you can try a micro controller and compile this code.

Arduino IDE example:

```
void setup() {
 Serial.begin(9600);
 pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
 delay(1000);
 Serial.println("OUT");
 while(Serial.available() > 0) {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);
  Serial.print("I received: ");
  Serial.println(Serial.read(), DEC);
 }
}
```

See also section 15.9 Device.USB.

68.1 OnUsbReadReady:

Syntax: OnUsbReadReady:

Label for an interrupt handler which traps arrival of USB data. When done, execute the **Usb.onReadReady.resume** statement to resume the interrupted program.

68.2 OnUsbStatus:

Syntax: OnUsbStatus:

Label for an interrupt handler which traps arrival of a status message via USB. When done, execute the **Usb.onStatus.resume** statement to resume the interrupted program.

68.3 Usb.close

Syntax: Usb.close

Closes the USB channel.

68.4 Usb.devices

Syntax: Usb.devices deviceId_Array[], deviceDescrip_Array$[]

Returns arrays with information on selectable USB devices. The first numeric array contains the device IDs, and the second string array contains device descriptions. Note that mass storage devices will be included.

68.5 Usb.onReadReady.resume

Syntax: Usb.onReadReady.resume

This statement should be placed at the end of the interrupt handler at **OnUsbReadReady:.**

68.6 Usb.onStatus.resume

Syntax: Usb.onStatus.resume

This statement should be placed at the end of the interrupt handler at **OnUsbStatus:.**

68.7 Usb.open

Syntax: Usb.open {<bundle_nexp>}

Opens a USB channel specified by the bundle <bundle_nexp>.

If more than one channel can be used for serial communication, the bundle key **_DeviceID** must be used to select one. See **Usb.devices**.

The bundle keys and possible values are in the table below:

Key	Type	Value
_DeviceID	numeric	Necessary if more than one channel can be used for serial communication. See Usb.devices.
_BaudRate	numeric	Sets the baud rate. Common are 30, 50, 75, 110, 130, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000, 230400, 250000, 256000, 500000, 512000, 921600, 1000000, 2000000 etc. in example. Keep in mind, that this BASIC! interpreter is limited in speed. Default is 115200 bits per second.
_DataBits	numeric	Sets the count of data bits. Available are 5, 6, 7 or 8. Default is 8.
_StopBits	numeric	Sets the count of stop bits. Available are 1, 1.5 or 2. Default is 1
_Parity	String	Sets the parity selectable to; _None, _Even, _Odd, _Mark and _Space. Default is _None.
_FlowControl	String	Sets flow control selectable are _Off, _Xon_Xoff (Maybe, currently not tested.), _Dsr_Dtr or _Rts_Cts. The last two are only for CP2102 and FT232. Default is _Off.
_Pause	numeric	Some Arduinos would need some sleep because firmware wait some time to know whether a new sketch is going to be uploaded or not. In this case are 2000 milliseconds a candidate for a first try. Default is 0.
_Delimiter	String	Different programs, microcontrollers, terminals etc. may send different codes for a new line. Usually a LF terminates a line or message. Some terminals use CR LF like CHR$(13) + CHR$(10). For control codes like CR, LF or TAB(9)

Key	Type	Value
		you should use the CHR$() function. Default is CHR$(10) (LF).
_CharSet	String	If you need to transfer characters from 0 to 255, maybe for binary data, specify the "_ISO-8859-1" character set. This bundle key can also be changed during data transfer operations. Default is "_UTF-8".
_Base64	numeric	The data transfer will be encoded by the Bas64 encoding algorithm if the numeric argument is greater than 0. Both character sets handle Bas64 correctly. This bundle key can be changed during data transfer operations also. Default is 0.

68.8 Usb.read.bytes

Syntax: Usb.read.bytes <sval>

Reads a string of bytes via USB. Depending on the buffer size and other circumstances, the number of bytes is limited. An **<sval>** delimiter like the default Linefeed character CHR$(10) ("\n") initiates a "Have data to read!" message. This message triggers an interrupt detectable by **onUsbReadReady:**. Note, If the delimiter is "", each packet can be read as is.

Example:

```
Usb.devices d[], i$[]
Array.length al, d[]
If al > 1 & d[1] > -1 Then % If more than one device can be selected
 sel = 0
 Dialog.single retBut, sel, i$[], "Serial Devices", "OK", "CHANCEL"
EndIf

Bundle.put MB, "_BaudRate", 9600

If sel > 0 & retBut = 1 Then Bundle.Put MB, "_DeviceID", d[sel] : ?d[sel]

Usb.open mB
Do
 Pause 100
Until 0

onUsbReadReady:
Usb.read.bytes b$
b$ = Replace$(b$, Chr$(13), "") // Removes each Carriage Return if any.
Print B$
Usb.write "Test"
Usb.onreadready.resume
```

68.9 Usb.status

Syntax: Usb.status <svar>

Returns the current status message.

Possible are:

- A USB device is attached
- USB service started
- USB Ready
- USB Permission granted

- CTS change
- DSR change
- USB Permission not granted
- USB not supported
- No USB device connected
- USB device not working
- CDC driver not working
- USB device not supported
- Current USB device detached. To reconnect restart by RUN()
- Another USB device is detached

68.10 Usb.write

Syntax: Usb.write {<ok_lvar, }<sexp>

Writes an UTF-8 encoded String via USB by default. The data transfer can also be Base64 encoded or by the ISO-8859-1 character set.

The optional <ok_lvar> will contain 1 if the command was executed after a **USB ready** status. Otherwise it will contain 0.

69 User-Defined Functions

User-Defined Functions are BASIC! functions like **Abs**(n), **Mod**(a, b) and **Left$**(a$, n) except that the operation of the function is defined by the user.

User-Defined Functions may call other User-Defined Functions. A function can even recursively call itself.

You may define a function with the same name as a built-in function. The User-Defined Function always overrides the built-in function, and the built-in function is not accessible.

Each time a function is called from another function a certain amount of memory is used for the execution stack. The depth of these nested calls is limited only by the amount of memory that your particular Android device allocates to applications.

69.1 Variable Scope

All variables created while running a User-Defined Function are private to the function. A variable named v$ in the main program is not the same as variable v$ within a function. Furthermore, a variable named v$ in a recursively called function is not the same v$ in the calling function.

A function cannot access variables created outside of the function except as parameters passed by reference. See **Fn.def**, below, for an explanation of parameters passed by value and by reference.

All variables created while running a User-Defined Function are destroyed when the function returns. When an array variable is destroyed, its storage is reclaimed. However, when a data structure pointer is destroyed, the data structure is not destroyed (see next section).

69.2 Data Structures in User-Defined Functions

Data structures (List, Stack, Bundle, bitmap, graphical object – anything referenced through a pointer) are global in scope. That is, if a variable is used as a pointer to a data structure, it points to the same data structure whether it is used inside or outside of a function. The data structure may have been created in the main program, the same user-defined function, or some other user-defined function.

This means that if you pass a pointer to a bundle, for example, and modify that bundle inside the function, the changes will be retained when the function returns. It also means that a function can modify graphical objects created outside of the function.

Data structures (List, Stack, Bundle, or graphical object) created while running a User-Defined Function are not destroyed when the function returns. Local variables that point to the data structures are lost, but you can return a data structure pointer as the function's return value or through a parameter passed by reference.

69.3 Call

Syntax: Call <user_defined_function>

Executes the user-defined function. Any value returned by the function will be discarded.

The **Call** command keyword is optional. Just as BASIC! can infer the **Let** command from a line that starts with a variable, it can infer the **Call** command from a line that starts with a function name.

For example, if you have defined a function like this:

```
Fn.def MyFunction(x, y$, z)
 < your code here >
Fn.end
```

You can execute the function, ignoring its return value, with either of these statements:

```
Call MyFunction(a, b$, c)
MyFunction(a, b$, c)
```

As with **Let**, you must use **Call** if your function name starts with a BASIC! command keyword. It is also a little faster to execute a function with **Call** than to make BASIC! infer the command. See **Let** for details.

69.4 Fn.def

Syntax: Fn.def name|name$({nvar}|{svar}|Array[]|Array$[], ... {nvar}|{svar}|Array[]|Array$[]){[]}

Begins the definition of a function. This command names the function and lists the parameters, if any.

If the function name ends with the $ character then the function will return a string, otherwise it will return a number. If it ends with [], the function returns a string array or numeric array. The parameter list can contain as many parameters as needed, or none at all. The parameters may be numeric or string, scalar or array.

Your program must execute **Fn.def** before it tries to call the named function. Your program must not attempt to create more than one function with the same name, or the same function more than once. However, you may override a built-in function by defining your own function with the same name.

The following are all valid:

```
Fn.def cut$(a$, left, right)
Fn.def sum(a, b, c, d, e, f, g, h, i, j)
Fn.def sort(v$[], direction)
Fn.def pi()               % Overrides built-in. You can make π = 3!
```

Parameters create variables visible only inside the function. They can be used like other variables created inside the function (see Variable Scope).

There are two types of parameters: call by reference and call by value. Call by value means that the calling variable value (or expression) is copied into the called variable. Changes made to the called variable within the function do not affect the value of the calling variable. Call by reference means that the calling variable value is changed if the called variable value is changed within the function.

Scalar (non-array) function variables can be either call by value or call by reference. Which type the variable will be depends upon how it is called. If the calling variable has the "&" character in front of it, then the variable is call by reference. If there is no "&" in front of the calling variable name then the variable is call by value.

```
Fn.def test(a)
 a = 9
 Fn.rtn a
Fn.end

a = 1
Print test(a), a   % will print: 9, 1
Print test(&a), a  % will print: 9, 9
```

Array parameters are always called by reference.

```
Fn.def test(a[])
 a[1] = 9
 Fn.rtn a[1]
Fn.end

Dim a[1]
```

```
a[1] = 1
Print test(a[]), a[1]  % prints: 9, 9
```

Along with the function's return value, you can use parameters passed by reference to return information to a function's caller.

69.5 Fn.end

Syntax: Fn.end

Ends the definition of a user-defined function. Every function definition must end with **Fn.end**.

When your function is running, executing the **Fn.end** statement causes the function to terminate and return a default value. If the function type is numeric then the default return value is 0.0. A string function returns the empty string (""). An array function returns a one-dimensional array with one element (containing 0.0 or "").

69.6 Fn.rtn

Syntax: Fn.rtn <sexp>|<nexp>{[]}

Causes the function to terminate execution and return the value of the return expression <sexp>|<nexp>, or an array. The return expression type, string or number, must match the type of the function name. **Fn.rtn** statements may appear anywhere in the function.

A function can return a single scalar value or an array. It cannot return a data structure (List, Stack, Bundle, or graphical object), but it can return a pointer to a data structure.

Note: You can also return information to a function's caller through parameters passed by reference.

69.7 Globals.all

Syntax: Globals.all

After this command executes, the scope of all variables in functions will be global**.**

With the commands **Locals.on** and **Locals.off**, you can change the scope of variables in a part of, or in the entire function to local.

Be very careful with this command! With some libraries (like GW.bas), you can get into trouble if you do not bracket the library calls with **Locals.on** and **Locals.off**. **The use inside of functions is strongly not recommended, because the results are wrong if the function is a part of a function call chain inside an interrupt routine.**

A better alternative is to use bundles. They are global, thus a global container with all the variable types is available. Be careful if you include a library like GW.bas. It uses bundle 1 as the global bundle pointer. So use **Include "GW.bas"** as the first line in your main program, and then use something like **Bundle.create myGlobals**. In this case myGlobals is 2.

Setting the global bundle pointer in **Fn.def** (myGlobals, …) makes your code readable.

69.8 Globals.none

Syntax: Globals.none

After this command executes, the scope of all variables in functions will be **local**, except for the variables that were imported by **Globals.fnImp**.

69.9 Globals.fnImp or Fn.import

Syntax: Globals.fnImp <varexp> {… , <varexp>}, …

or

Syntax: Fn.import <varexp> {… , <varexp>}, …

Imports variables from the main program area into a function. Thus you have access to the specified variables from the main area.

Globals.fnImp, **Globals.all** and **Globals.none** should never be used simultaneously in the same function.

69.10 Locals.on

Makes the variables in functions local. See `Globals on`.

69.11 Locals.off

Makes the variables in functions not local, so they can be global. See `Globals on`.

70 XML Commands

XML (E**x**tensible **M**arkup **L**anguage) is a text-based format for the exchange of structured information. These can be documents, configurations, books, invoices, and more. XML is not supported in all aspects, and is limited by the abilities of the JSON-XML and the XML-JSON parser. So only well formed expressions are parsed. For a well formed output add:

```
<?xml version="1.0" encoding="utf-8"?>
```

at the begin of the string.

Unfortunately, an additional DOCTYPE declaration (DTD) entry for a valid XML document is needed. See the documentation at https://www.w3schools.com/xml/default.asp.

70.1 Is_Xml

Syntax: Is_Xml (<xml_sexp>)

Checks if <Xml_sexp> is a parseable XML string.

Returns 1 if true, 0 if false.

See also: **XmlToJson$()**

70.2 JsonToXml$

Syntax: JsonToXml$ (<json_sexp>{{,<spaces_nexp>}, <shrink_nexp>})

Retruns an XML string converted from a JSON string. Needs a parseable XML string.

If the optional <spaces_nexp> has a number > -1, a structural printout is delivered. The number of spaces defines the tabulator distance.

If <spaces_nexp> is -1 or not given, a compact printout is returned.

If <shrink_nexp> greater than 0, <spaces_nexp> will be overwritten by 0 and the line feeds will be removed.

If an error occurred, an empty string will be returned.

Note, that

```
<example name:"Marc" age:31 male:true />
```

returns

```
<example><name>Marc</name><age>31</age><male>true</male></example>
```

if the optional <shrink_lexp> is switched to off by 0.

See also: **XmlToJson$()**

IV - APPENDICES

1 Supported Media Formats

From https://developer.android.com/guide/topics/media/platform/supported-formats.

This document describes the media codec, container, and network protocol support provided by the Android platform.

The tables below describe the media format support built into the Android platform. YES means the format is available on handhelds and tablets running all Android versions. Where a specific Android platform is specified, the format is available on handsets and tablets running that version and all later versions. The format might also be available in earlier versions, but this is not guaranteed. On form factors other than handsets and tablets, media format support may vary.

Note that a particular mobile device might support additional formats or file types that are not listed in these tables. In addition, if you use a Media Codec directly, you can access any of the available media formats regardless of the supported file types and container formats.

Audio Support				
Format	**Encoder**	**Decoder**	**Details**	**File Types Container Formats**
AAC LC	YES	YES	Support for mono/stereo/5.0/5.1 content with standard sampling rates from 8 to 48 kHz.	• 3GPP (.3gp) • MPEG-4 (.mp4, .m4a) • ADTS raw AAC (.aac, decode in Android 3.1+, encode in Android 4.0+, ADIF not supported) • MPEG-TS (.ts, not seekable, Android 3.0+)
HE-AACv1 (AAC+)	Android 4.1+	YES		
HE-AACv2 (enhanced AAC+)		YES	Support for stereo/5.0/5.1 content with standard sampling rates from 8 to 48 kHz.	
xHE-AAC		Android 9+	Support for up to 8ch content with standard sampling rates from 8 to 48 kHz	
AAC ELD (enhanced low delay AAC)	Android 4.1+	Android 4.1+	Support for mono/stereo content with standard sampling rates from 16 to 48 kHz	
AMR-NB	YES	YES	4.75 to 12.2 kbps sampled @ 8kHz	• 3GPP (.3gp) • AMR (.amr)
AMR-WB	YES	YES	9 rates from 6.60 kbit/s to 23.85 kbit/s sampled @ 16kHz	
FLAC	Android 4.1+	Android 3.1+	Mono/Stereo (no multichannel). Sample rates up to 48 kHz (but up to 44.1 kHz is recommended on devices with 44.1 kHz output, as the 48 to 44.1 kHz downsampler does not include a low-pass filter). 16-bit recommended; no dither applied for 24-bit.	• FLAC (.flac) • MPEG-4 (.mp4, .m4a, Android 10+)
MIDI		YES	MIDI Type 0 and 1. DLS Version 1 and 2. XMF and Mobile XMF. Support for ringtone formats RTTTL/RTX, OTA, and iMelody	• Type 0 and 1 (.mid, .xmf, .mxmf) • RTTTL/RTX (.rtttl, .rtx) • OTA (.ota) • iMelody (.imy)

Audio Support				
Format	**Encoder**	**Decoder**	**Details**	**File Types Container Formats**
MP3		YES	Mono/Stereo 8-320Kbps constant (CBR) or variable bit-rate (VBR)	• MP3 (.mp3) • MPEG-4 (.mp4, .m4a, Android 10+) • Matroska (.mkv, Android 10+)
Opus	Android 10+	Android 5.0+		• Ogg (.ogg) • Matroska (.mkv)
PCM/WAVE	Android 4.1+	YES	8- and 16-bit linear PCM (rates up to limit of hardware). Sampling rates for raw PCM recordings at 8000, 16000 and 44100 Hz.	WAVE (.wav)
Vorbis		YES		• Ogg (.ogg) • Matroska (.mkv, Android 4.0+) • MPEG-4 (.mp4, .m4a, Android 10+)

2 Color Table

Predefined colors based on official HTML Color Names of the World Wide Web Consortium (W3C).

Also allowed is the following syntax:

```
"_{<alpha>,} HSV<hue[0...360]>{{,<saturation [0...1]>}, <valueOfBrightness
[0...1]>}"
```

The string "_200,HSV300" returns the color blue with aplha = 200.

HTML Name	Code	Name	HTML Hex Code
AliceBlue	**255,**240,248,255	**_255,**AliceBlue	#**FF**F0F8FF
AliceBlue	240,248,255	_AliceBlue	#F0F8FF
AntiqueWhite	250,235,215	_AntiqueWhite	#FAEBD7
Aqua	0,255,255	_Aqua	#00FFFF
Aquamarine	127,255,212	_Aquamarine	#7FFFD4
Azure	240,255,255	_Azure	#F0FFFF
Beige	245,245,220	_Beige	#F5F5DC
Bisque	255,228,196	_Bisque	#FFE4C4
Black	0,0,0	_Black	#000000 or #000 or #F000
BlanchedAlmond	255,235,205	_BlanchedAlmond	#FFEBCD
Blue	0,0,255	_Blue	#0000FF
BlueViolet	138,43,226	_BlueViolet	#8A2BE2
Brown	165,42,42	_Brown	#A52A2A
BurlyWood	222,184,135	_BurlyWood	#DEB887
CadetBlue	95,158,160	_CadetBlue	#5F9EA0
Chartreuse	127,255,0	_Chartreuse	#7FFF00
Chocolate	210,105,30	_Chocolate	#D2691E
Coral	255,127,80	_Coral	#FF7F50
CornflowerBlue	100,149,237	_CornflowerBlue	#6495ED
Cornsilk	255,248,220	_Cornsilk	#FFF8DC
Crimson	220,20,60	_Crimson	#DC143C
Cyan	0,255,255	_Cyan	#00FFFF
DarkBlue	0,0,139	_DarkBlue	#00008B
DarkCyan	0,139,139	_DarkCyan	#008B8B
DarkGoldenRod	184,134,11	_DarkGoldenRod	#B8860B
DarkGray	169,169,169	_DarkGray	#A9A9A9
DarkGreen	0,100,0	_DarkGreen	#006400
DarkKhaki	189,183,107	_DarkKhaki	#BDB76B
DarkMagenta	139,0,139	_DarkMagenta	#8B008B
DarkOliveGreen	85,107,47	_DarkOliveGreen	#556B2F
DarkOrange	255,140,0	_DarkOrange	#FF8C00

HTML Name	Code	Name	HTML Hex Code
DarkOrchid	153,50,204	_DarkOrchid	#9932CC
DarkRed	139,0,0	_DarkRed	#8B0000
DarkSalmon	233,150,122	_DarkSalmon	#E9967A
DarkSeaGreen	143,188,143	_DarkSeaGreen	#8FBC8F
DarkSlateBlue	72,61,139	_DarkSlateBlue	#483D8B
DarkSlateGray	47,79,79	_DarkSlateGray	#2F4F4F
DarkTurquoise	0,206,209	_DarkTurquoise	#00CED1
DarkViolet	148,0,211	_DarkViolet	#9400D3
DeepPink	255,20,147	_DeepPink	#FF1493
DeepSkyBlue	0,191,255	_DeepSkyBlue	#00BFFF
DimGray	105,105,105	_DimGray	#696969
DodgerBlue	30,144,255	_DodgerBlue	#1E90FF
FireBrick	178,34,34	_FireBrick	#B22222
FloralWhite	255,250,240	_FloralWhite	#FFFAF0
ForestGreen	34,139,34	_ForestGreen	#228B22
Fuchsia	255,0,255	_Fuchsia	#FF00FF
Gainsboro	220,220,220	_Gainsboro	#DCDCDC
GhostWhite	248,248,255	_GhostWhite	#F8F8FF
Gold	255,215,0	_Gold	#FFD700
GoldenRod	218,165,32	_GoldenRod	#DAA520
Gray	128,128,128	_Gray	#808080
Green	0,128,0	_Green	#008000
GreenYellow	173,255,47	_GreenYellow	#ADFF2F
HoneyDew	240,255,240	_HoneyDew	#F0FFF0
HotPink	255,105,180	_HotPink	#FF69B4
IndianRed	205,92,92	_IndianRed	#CD5C5C
Indigo	75,0,130	_Indigo	#4B0082
Ivory	255,255,240	_Ivory	#FFFFF0
Khaki	240,230,140	_Khaki	#F0E68C
Lavender	230,230,250	_Lavender	#E6E6FA
LavenderBlush	255,240,245	_LavenderBlush	#FFF0F5
LawnGreen	124,252,0	_LawnGreen	#7CFC00
LemonChiffon	255,250,205	_LemonChiffon	#FFFACD
LightBlue	173,216,230	_LightBlue	#ADD8E6
LightCoral	240,128,128	_LightCoral	#F08080
LightCyan	224,255,255	_LightCyan	#E0FFFF
LightGoldenRodYellow	250,250,210	_LightGoldenRodYellow	#FAFAD2
LightGray	211,211,211	_LightGray	#D3D3D3
LightGreen	144,238,144	_LightGreen	#90EE90
LightPink	255,182,193	_LightPink	#FFB6C1
LightSalmon	255,160,122	_LightSalmon	#FFA07A
LightSeaGreen	32,178,170	_LightSeaGreen	#20B2AA
LightSkyBlue	135,206,250	_LightSkyBlue	#87CEFA

HTML Name	Code	Name	HTML Hex Code
LightSlateGray	119,136,153	_LightSlateGray	#778899
LightSteelBlue	176,196,222	_LightSteelBlue	#B0C4DE
LightYellow	255,255,224	_LightYellow	#FFFFE0
Lime	0,255,0	_Lime	#00FF00
LimeGreen	50,205,50	_LimeGreen	#32CD32
Linen	250,240,230	_Linen	#FAF0E6
Magenta	255,0,255	_Magenta	#FF00FF
Maroon	128,0,0	_Maroon	#800000
MediumAquaMarine	102,205,170	_MediumAquaMarine	#66CDAA
MediumBlue	0,0,205	_MediumBlue	#0000CD
MediumOrchid	186,85,211	_MediumOrchid	#BA55D3
MediumPurple	147,112,219	_MediumPurple	#9370DB
MediumSeaGreen	60,179,113	_MediumSeaGreen	#3CB371
MediumSlateBlue	123,104,238	_MediumSlateBlue	#7B68EE
MediumSpringGreen	0,250,154	_MediumSpringGreen	#00FA9A
MediumTurquoise	72,209,204	_MediumTurquoise	#48D1CC
MediumVioletRed	199,21,133	_MediumVioletRed	#C71585
MidnightBlue	25,25,112	_MidnightBlue	#191970
MintCream	245,255,250	_MintCream	#F5FFFA
MistyRose	255,228,225	_MistyRose	#FFE4E1
Moccasin	255,228,181	_Moccasin	#FFE4B5
NavajoWhite	255,222,173	_NavajoWhite	#FFDEAD
Navy	0,0,128	_Navy	#000080
OldLace	253,245,230	_OldLace	#FDF5E6
Olive	128,128,0	_Olive	#808000
OliveDrab	107,142,35	_OliveDrab	#6B8E23
Orange	255,165,0	_Orange	#FFA500
OrangeRed	255,69,0	_OrangeRed	#FF4500
Orchid	218,112,214	_Orchid	#DA70D6
PaleGoldenRod	238,232,170	_PaleGoldenRod	#EEE8AA
PaleGreen	152,251,152	_PaleGreen	#98FB98
PaleTurquoise	175,238,238	_PaleTurquoise	#AFEEEE
PaleVioletRed	219,112,147	_PaleVioletRed	#DB7093
PapayaWhip	255,239,213	_PapayaWhip	#FFEFD5
PeachPuff	255,218,185	_PeachPuff	#FFDAB9
Peru	205,133,63	_Peru	#CD853F
Pink	255,192,203	_Pink	#FFC0CB
Plum	221,160,221	_Plum	#DDA0DD
PowderBlue	176,224,230	_PowderBlue	#B0E0E6
Purple	128,0,128	_Purple	#800080
RebeccaPurple	102,51,153	_RebeccaPurple	#663399
Red	255,0,0	_Red	#FF0000
RosyBrown	188,143,143	_RosyBrown	#BC8F8F

HTML Name	Code	Name	HTML Hex Code
RoyalBlue	65,105,225	_RoyalBlue	#4169E1
SaddleBrown	139,69,19	_SaddleBrown	#8B4513
Salmon	250,128,114	_Salmon	#FA8072
SandyBrown	244,164,96	_SandyBrown	#F4A460
SeaGreen	46,139,87	_SeaGreen	#2E8B57
SeaShell	255,245,238	_SeaShell	#FFF5EE
Sienna	160,82,45	_Sienna	#A0522D
Silver	192,192,192	_Silver	#C0C0C0
SkyBlue	135,206,235	_SkyBlue	#87CEEB
SlateBlue	106,90,205	_SlateBlue	#6A5ACD
SlateGrey	112,128,144	_SlateGrey	#708090
Snow	255,250,250	_Snow	#FFFAFA
SpringGreen	0,255,127	_SpringGreen	#00FF7F
SteelBlue	70,130,180	_SteelBlue	#4682B4
Tan	210,180,140	_Tan	#D2B48C
Teal	0,128,128	_Teal	#008080
Thistle	216,191,216	_Thistle	#D8BFD8
Tomato	255,99,71	_Tomato	#FF6347
Turquoise	64,224,208	_Turquoise	#40E0D0
Violet	238,130,238	_Violet	#EE82EE
Wheat	245,222,179	_Wheat	#F5DEB3
White	255,255,255	_White	#FFFFFF
WhiteSmoke	245,245,245	_WhiteSmoke	#F5F5F5
Yellow	255,255,0	_Yellow	#FFFF00
YellowGreen	154,205,50	_YellowGreen	#9ACD32

3 Miscellaneous

3.1 Something about AndoidManifest.xml

If you want only encoded IP connections like Https instead of Http change android:usesCleartextTraffic to false.

3.2 Something about touch events

Android triggers touch events at a distance of at least 16 to 18 milliseconds.

Standard Values Some of them can be changed by the user in global preferences. All device independent pixels (DIPs) are in conjunction to the Android standard 160 DPI resolution.		
Tap Timeout	100	Duration in milliseconds we will wait to see if a touch event is a tap or a scroll. If the user does not move within this interval, it is considered to be a tap.
Double Tap Timeout	300	Duration in milliseconds between the first tap's up event and the second tap's down event for an interaction to be considered a double-tap.
Long Press Timeout	400	Duration in milliseconds before a press turns into a long press
Maximum Fling Velocity	8000	Maximum velocity to initiate a fling, as measured in DIPs per second.
Minimum Fling Velocity	50	Minimum velocity to initiate a fling, as measured in DIPs per second.
Touch Slop	8	Distance in DIPs a touch can wander before we think the user is scrolling

3.3 NaN (Not a Number) or Infinity values (IEEE 754)

These are floating point values in our case from type Double.
If you try SQR (-1) or 0/0 you get a NaN, because it is an undefined operation.
But if you try 1/0 you get an Infinity, because it is an infinite result.
Or you get these values per definition like VAL("Infinity"), VAL("-Infinity") and VAL("NaN").

This differs from older Basic implementations and in some parts also from RFO-Basic 1.91.
That should not have a negative impact on older code. Hopefully in this development stage, you will only get a runtime error if you want to handle these values and functions, where input values must be of the type Integer or Long.
Functions which use BigDecimal inside do not deal with NaN (Not a Number) or Infinity values.
Examples: BigD(decimal) command group, List.join
Functions which use <lexp> and <nexp> interpret these values as 0.
For detection use Is_NaN and Is_Infinite. The last one returns -1 if the value is negative.

!NaN Example

```
Dim x[1]
x[1] = 1
Array.std_dev nanV, x[ ] % Returns NaN in all Basic! versions.
If nanV Then ? "true" : Else ? "false"  % → true   But that is wrong, see below
If Is_Number("NaN") Then ? "true" : Else ? "false"  % → true
If Is_NaN(nanV) Then ? "true" : Else ? "false"  % → true
If nanV = nanV Then ? "true" : Else ? "false"  % → false
If nanV <> nanV Then ? "true" : Else ? "false"  % → true
If nanV < 0 Then ? "true" : Else ? "false"  % → false
If nanV > 0 Then ? "true" : Else ? "false"  % → false
```

!Infinity Example

```
!infiV = -1/0             % Returns -Infinity.
infiV = Val("-Infinity")  % Returns -Infinity in all Basic! versions.
? infiV
If infiV Then ? "true" : Else ? "false"  % → true
If Is_Number("-Infinity") Then ? "true" : Else ? "false"  % → true
If Is_Infinite(infiV) Then ? "true" : Else ? "false"  % → true
?  Is_Infinite(infiV)                          % → -1
If infiV = infiV Then ? "true" : Else ? "false"  % → true
If infiV <> infiV Then ? "true" : Else ? "false"  % → false
If infiV < 0 Then ? "true" : Else ? "false"  % → true
If infiV > 0 Then ? "true" : Else ? "false"  % → false
```

Use instead of:

```
If nanV Then ? "true" : Else ? "false"  % → true   But that is wrong
If !Is_NaN(nanV) & nanV Then ? "true" : Else ? "false"  % → false   That is
right
While nanV % → true   But that is wrong
While !Is_NaN(nanV) & nanV  % → false   That is right
Until nanV % → true   But that is wrong
Until !Is_NaN(nanV) & nanV  % → false   That is right
```

Sum of Arrays

```
nanV = Val("NaN")
? nanV
If Is_NAN(nanV) Then ? "true" : Else ? "false"  % → true
infiV = Val("-Infinity") % Returns -Infinity in all Basic! versions.
? infiV
If infiV Then ? "true" : Else ? "false"  % → true

Array.load exampl[], 1, 2, 3
Array.sum s, exampl[]
Print s  % → 6

Array.load exampl[], 1, 2, 3, nanV
Array.sum s, exampl[]
Print s  % → NaN

Array.load exampl[], 1, 2, 3, infiV
Array.sum s, exampl[]
Print s  % → -Infinity

Array.load exampl[], 1, 2, 3, nanV, infiV
Array.sum s, exampl[]
Print s  % → NaN
```

Comparison with NaN

A comparison with a NaN always returns an *unordered result* even when comparing with itself. The comparison predicates are either signaling or non-signaling on quiet NaN operands; the signaling versions signal the invalid operation exception for such comparisons. The equality and inequality predicates are non-signaling so $x = x$ returning false can be used to test if x is a quiet NaN. The other standard comparison predicates are all signaling if they receive a NaN operand, the standard also provides non-signaling versions of these other predicates. The predicate *isNaN(x)* determines if a value is a NaN and never signals an exception, even if x is a signaling NaN.

Comparison between NaN and any floating-point value *x* (including NaN and ±∞)						
Comparison	**NaN ≥ *x***	**NaN ≤ *x***	**NaN > *x***	**NaN < *x***	**NaN = *x***	**NaN ≠ *x***
Result	Always **False**	Always **False**	Always **False**	Always **False**	Always **False**	Always **True**

See also
https://docs.oracle.com/cd/E19957-01/806-3568/ncg_intro.html#110
https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

V - ALPHABETICAL INDEX OF COMMANDS AND FUNCTIONS

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

!

?

*

/

%