

APPENDIX TO THE MANUAL

(De Re BASIC! Version 1.91, 2017-03-15)

ENHANCEMENTS and CORRECTIONS

A Test Version called OliBasicXV based on BASIC! 1.91 2017-03-18gt

Formatted like the manual, changes and corrections colored in **dark Magenta**.

What should you keep in mind, if you use source code that runs with BASIC! 1.91 perfectly.

1. The minSDK is 8, the targetSDK is 19
2. Look at FILE.EXISTS the second parameter is now a value not a String expression!
3. If you use GW.bas from Nicolas use a version > 4.6, because **the** sentence **before**.
4. The automatic low memory **warning** is switched off. Use instead OnLowMemory:.
5. The Global Value Backdoor, after an interrupt is trapped, is now closed.

See also **Abe** maked sentences.

To prevent issues in older code use:

ON***:

GLOBALS.ALL

....

GLOBALS.NONE

***.RESUME

6. The RUN command has new possibilities.
7. AndroidManifest.xml has the new permission ACCESS_NETWORK_STATE.
8. This version use absolute paths instead of relative paths.
9. An internal directory is created at the first start.
10. Add more system constants at FILES.ROOT
11. FILE.EXISTS returns readable and writeable infos, too.

Thanks to Alberto, Janusz, Mog, Nicolas, Emile, Roy for testing and support.
Special thanks to Emile for text corrections.

Happy coding
Gregor Tiemeyer

Gregor's Stuff

App.installed <flag_nexp>,

<package_sexp>{{{,<versionName_sexp>},<versionCode_nexp>},<pmRaw_sexp>}

If the android package is installed, flag_nexp returns 1, else 0. Insert the package id like "com.rfo.basic" or "all.sub.My.APP". VersionName and versionCode are results from the APK manifest. Package manager returns a raw string list in pmRaw_sexp.

App.load <package_sexp>

Loads and installs an android app package (*.apk) with a given parse-able URI.

Example:

```

File.root dataPath$
! Can be parsed:
Package$ = "file://" + dataPath$ + "/" + "Bookworm_de.apk"
Package$ = "market://details?id=" + "com.opera.mini.native"
! Can not be parsed directly:
!!
Package$ = "http://mougino.free.fr/tmp/920EditorforBASIC_13720.apk"
(Download the Apk first to the data path. See also the example which
can be found in the Sample Program file, f00a_download_manual.bas.)
Package$ = "file://" + "/android_asset/" + "Bookworm_en.apk"
(Protected, copy the Apk first to the data path.)
!!
App.Load Package$

```

APP.SAR <pointer_nexp>

Start And Receive an App or Broadcast with a Java like Interface.

The references are cut!

Eg. FileBrowser, BarCodeScanner, GoogleMaps, ... are supported.

Note: The Blackmoon FileBrowser is suggested, because it does not need the clipboard for multiple files.

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();" ~
"EOCL"

BUNDLE.CREATE appVarPointer
BUNDLE.PL appVarPointer, "_CommandList",commandListPointer
BUNDLE.PL appVarPointer, "_ResultList",resultListPointer
BUNDLE.PUT appVarPointer,"theFilePath$","No file selected!"

APP.SAR appVarPointer

```

ATTENTION: In Java variable, class and function names are case sensitive. In most Java examples → **intent**. ← is a variable of the class Intent and have to be deleted.

Instead → **Intent**. ← is the class and strongly not to delete.

_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.

Basic! Variables used in the CommandList (only the bracketed variables with exclamation like "!"variable\$!") and ResultList have to be put in the transfer bundle, too.

Ignore and delete → this. ←.

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

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

Supported:

- new Intent
- setAction
- setPackage
- putExtra only Bundles, Strings and the arguments true and false
- addCategory
- addFlags
- setData
- setDataAndType
- setType

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

This list will maybe expanded. Stay tuned!

Bundles

A Bundle is a group of values collected together into a single object. A bundle object may contain any number of string, numeric values [and also there arrays, lists, stacks, bundles 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 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.copy <SourcePointer_nexp>, <DestinationPointer_nexp>

Copies the source bundle to the destination Bundle. The references are cut!

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. The <type_svar> will contain an uppercase "S" if the type is a string. [For arrays the dimensions are added like S\[6\] or N\[2,2,5,..\].](#) The other types result in "List", "Stack" and "Bundle". If the bundle does not exist or does not contain the requested key, the command generates a run-time error.

Example:

```
BUNDLE.TYPE bptr, "age", type$  
PRINT type$ % will print N, see also Bundle.put
```

Bundle.put <pointer_nexp>, <key_sexp>, <value_nexp>|**Array[]**|<value_sexp>|**Array\$[]**

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

Example:

```
BUNDLE.PUT bptr, "first_name", "Frank"  
BUNDLE.PUT bptr, "age", 44
```

Bundle.pl <pointer_nexp>, <key_sexp>, <list_ptr_nexp>

Puts a list in a bundle. Read Bundle.Put.List instead Bundle.PL

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 a list pointer.

Example:

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

Bundle.ps <pointer_nexp>, <key_sexp>, <stack_ptr_nexp>

Puts a stack in a bundle. Read Bundle.Put.Stack instead Bundle.PS

The stack 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 stack pointer.

Example:

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

Bundle.pb <pointer_nexp>, <key_sexp>, <bundle_pointer_nexp>

Puts a bundle in a bundle. Read Bundle.Put.Bundle instead Bundle.PB

The bundle 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 references are cut!

Example:

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

Bundle.pp <pointer_nexp>, <key_sexp>, <bitmap_pointer_nexp>

Puts a picture in a bundle. Read BUNDLE.PUT.PICTURE instead Bundle.PP

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. The references are cut!

Example:

```
BUNDLE.PL bptr, "picture1", bitmapPtr
```

Bundle.get <pointer_nexp>, <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 (array, 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  
BUNDLE.GET bptr,"professions", professions$[]
```

Bundle.gl <pointer_nexp>, <key_sexp>, <list_ptr_nexp>

Read Bundle.Get.List instead Bundle.GL

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.GET bptr,"names", listPtr
```

Bundle.gs <pointer_nexp>, <key_sexp>, <stack_ptr_nexp>

Read Bundle.Get.Stack instead Bundle.GS

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.GET bptr,"toDos", stackPtr
```

Bundle.gb <pointer_nexp>, <key_sexp>, <bundle_ptr_nexp>

Read Bundle.Get.Bundle instead Bundle.GP

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

Example:

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

Bundle.gp <pointer_nexp>, <key_sexp>, <picture_ptr_nexp>

Read Bundle.Get.Picture instead Bundle.GP

Places the picture 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:

```
BUNDLE.GET bptr,"Picture2", bitmapPtr
```

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 Extra string from the Intent. The retBundleIndex returns a bundle. If no data is broadcasting retData returns "" and retBundleIndex returns an empty Bundle.

The returned Bundle with the retBundleIndex pointer contains the received Extras, so you have not to parse the retData String for parameters.

Bundle.save <pointer_nexp>, <fileName_sexp>

Saves a bundle into a file. Only recommended for temporary use, because the internal coding is operating system version depended.

Bitmaps in bundles are also not supported in this case.

Maybe there is a size limitation. Not tested yet.

Example:

```
BUNDLE.SAVE bptr, "saveState.bun"
```

Bundle.load <pointer_nexp>, <fileName_sexp>

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

Bitmaps in bundles are also not supported in this case.

Maybe there is a size limitation. Not tested yet.

Example:

```
BUNDLE.LOAD bptr, "saveState.bun"
```

Email.send <recipient_sexp>|Array\$[], <subject_sexp>, <body_sexp>~ {{{,<sendTo_sexp>},<cC_sexp>|Array\$[]},<bCC_sexp>|Array\$[]},~ <attachment_sexp>|Array\$[]}

The email message in the Body string expression will be sent to the named recipient(s) with the named subject heading.

<sendTo_sexp> specifies an app which should send. (Ex.: Gmail, WhatsApp, DropBox or OneDrive. See also App.installed.) <cC_sexp> and <bCC_sexp> place the CC and BCC recipient(s) in the mail header. <attachment_sexp> specifies the attachment(s).

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$[]
```

Interrupt Labels (Event Handlers)

You can perform physical actions that tell your BASIC! 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* command 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 may be called an *Interrupt Service Routine (ISR)* or, if you prefer, an *Event Handler*.

~~Attention: Also during that time, all variables are global. See LOCALS.ON and LOCALS.OFF in this regard.~~

When BASIC! executes the event's Resume command, 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.

An Interrupt Label looks and behaves just like any other label in BASIC!. However, you must not execute any of the Resume commands except to finish an event's handler.

All Interrupt Labels

BASIC! supports trapping of the following events. These interrupt labels and their Resume commands are described in various parts of this manual.

- **OnBackground:** **Background.resume**
- **OnBackKey:** **Back.resume**
- **OnBroadcast:** **Broadcast.resume**
- **OnBtReadReady:** **Bt.onReadReady.resume**
- **OnConsoleTouch:** **ConsoleTouch.resume**
- **OnError:** None (not a true event, see **OnError:**, below)
- **OnGrTouch:** **Gr.onGrTouch.resume**
- **OnKbChange:** **Kb.resume**
- **OnKeyDown:** **KeyDown.resume**
- **OnKeyPress:** **Key.resume**
- **OnLowMemory:** **LowMemory.resume**
- **OnMenuKey:** **MenuKey.resume**
- **OnTimer:** **Timer.resume**

OnBroadcast:

Interrupt label that traps on receive a broadcast. BASIC! executes the statements following the **OnBroadcast:** label until it reaches a **Broadcast.resume**.

Broadcast.resume

Resumes execution at the point in the BASIC! program where the **OnBroadcast:** interrupt occurred.

OnKeyDown:

Interrupt label that traps a tap_down on any key except the volume keys. They have to be switched on with VolKeys.On. BASIC! executes the statements following the

OnKeyDown: label until it reaches a **KeyDown.resume**.

KeyDown.resume

Resumes execution at the point in the BASIC! program where the **OnKeyPress:** interrupt occurred.

OnKeyPress:

Interrupt label that traps a tap_up on any key. BASIC! executes the statements following the **OnKeyPress:** label until it reaches a **Key.resume**. (**OnKeyUp** would describe it better.)

Key.resume

Resumes execution at the point in the BASIC! program where the **OnKeyPress:** interrupt occurred.

Locals.on

After this command the variables in functions are definitely **local**.

Locals.off

After this command the variables in functions can be **global**, if **Globals.all** is used.

~~After this command the variables in functions are only LOCAL if BASIC!'s interrupt stack is empty.~~

Globals.all

After this command all variables in functions are **global**.

~~Like an ON*: label.~~

Be very carefully with this command! With the **Include** command and some libs like GW.bas you will get in trouble if you do not put the lib calls, **Locals.on** and **Locals.off**, in brackets.

A good alternative option is to rather use bundles. They are global, thus a global container with all the variable types is available.

Globals.none

After this command the variables in functions are **local** ~~if BASIC!'s interrupt stack is empty.~~

~~Like a "RESUME" command.~~

Broadcasts

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 BASIC! (different package IDs are needed) at runtime, maybe use this action addresses:

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

Broadcast.init <action_sexp> | Array\$[]

Initializes a Broadcast Receiver

OnBroadcast:

Interrupt label that traps a received broadcast. BASIC! executes the statements following the **OnBroadcast:** label until it reaches a **Broadcast.resume**.

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

Receives a Broadcast message (Intent) according to the recAction action address. With retData you get the Data Extra string from the Intent. The retBundleIndex returns a bundle. If no data is broadcasting retData returns "" and retBundleIndex returns an empty Bundle.

Broadcast.resume

Resumes execution at the point in the BASIC! program where the **OnBroadcast:** interrupt occurred.

Broadcast.close

Closes the Broadcast Receiver

Broadcast.bundle <sndAction_sexp>, <key_sexp>, <bundle_ptr_nvar>

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

Broadcast.string <sndAction_sexp>, <key_sexp>, <msg_sexp>

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

Inkey\$ <svar>{, <rawKeyEvent_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 process.

With rawKeyEvent you get an optional raw key event description with action, keyCode, scanCode, metaState, flags, repeatCount, eventTime, downTime, deviceId and source values.

Keydown.on

Key Down Detection possible.

Keydown.off

No Key Down Detection. Default status.

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 (mod the sum of all of the intervals).

The intervals list controls the length of the dashes.

The GR.set.stroke controls the thickness of the dashes.

Note: This path effect only affects drawing with 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).

Example:

```
scale = 2
LIST.CREATE N, pathPatternType1
LIST.ADD pathPatternType1, 10*scale, 7*scale, 3*scale, 7*scale
GR.SET.DASHPATHEFFECT pathPatternType1, 5*scale
```

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), "_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" and "y".

Example:

```
GR.COLOR 255,255,255,255,0
LIST.CREATE S, pathDraft1
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
!!
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
!!
"_LineTo"~
STR$(cPx-R4*COS(Angle2*PI()/180)), STR$(cPy+R4*SIN(Angle2*PI()/180))~ %Point 2
!!
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
!!
```

```

"_LineTo"~
STR$(cPx-R4),STR$(cPy)~ %Point 2
!!
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
!!
"_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
!!
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
!!
"_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
!!
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 the arc.
StartAngle                Starting angle (in degrees) where the arc begins
SweepAngle                Sweep angle (in degrees) measured clockwise(!).
!!
"_ArcTo"~
STR$(cPx),STR$(cPy-R4/4)~ %Left, Top
STR$(cPx+R4),STR$(cPy+R4/4)~ %Right, Bottom
STR$(0),STR$(180)~ %StartAngle, SweepAngle
!!
Close the current contour. If the current point is not equal to the first point of the contour,
a line segment is automatically added.
!!
!="_Close"~
!!
After _End the following items are ignored.
!!
"_End"
GR.SET.DASHPATHEFFECT pathPatternType1, 5*scale
!!
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
!!
GR.PATH gfirst, pathDraft1, -50, 0

```

GR.RENDER

Gr.rect <obj_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 **Display** 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", "bottom", "rx" and "ry".

Audio.info <aft_nvar>, <bundle_pointer_nvar>

Returns a bundle with the system value keys:

_Album, _Artist, _Title, _BitRate, _Duration, _LocationPath, _StreamMetadata

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

Loads a music file **or 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.

The file must be in the "<pref base drive>/ref-basic/data/" directories or one of its subdirectories.

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 != 0
  AUDIO.LOAD aft, fn$
  IF aft != 0
    AUDIO.STOP
    AUDIO.PLAY aft
  ENDIF
ENDIF
...
AUDIO.LOAD aft, "http://amp1.cesnet.cz:8000/cro1.ogg"
```

Audio.record.peak <level_nvar>

With this **PEAK** command you get with <level_nvar> the raw max. amplitude value between to calls.

The **Audio.record.START** command initializes the **PEAK** process for the first time.

If no current recording process available, **PEAK** returns -1.

TTS.kill

TTS.kill stops the **TTS** process without waiting for finish. With this command you have to code carefully! You have to be guaranteed, that all following commands including until **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 run **TTS.init** again.

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").

File Handling

A new feature is the ability to cross the directory borders of BASIC! with the URI start "file://".

Example:

```
FILE.ROOT dataPath$, "_Dcim"  
fn$ = "file://" + dataPath$
```

About Android Directories

A distinction is made between **internal** and **external** directories.

Internal directories are owned by the App, their indicator is the App's package name (e.g. com.rfo.basic) in the absolute path. If the App is be deinstalled **all** the **internal** directories are deleted, too. The exceptions are **internal** directories upon **removable** SD cards before Android 4.4 KitKat (API19). This version creates a **private internal** directory at first start.

External directories are member of the **public** part of Android's file system. From the view of the app it is controlled **externally**. So you need permissions finally in the file AndroidManifest.xml to access **external** directories. Please take a look at the compiler options, if you want to create an App.

The terms sd-ext or extSdcard are a case for a misunderstanding. These are expressions for **removable** SD cards.

Removable SD cards or USB sticks are normally **external** directories, too.

A bit odd in conjunction with this logic is the possibility to create an **internal** directory upon an **external** directory and sometimes also on a **removable** data carrier.

If you install the BASIC! Interpreter you get an **external public** directory rfo-basic.

If you deinstall BASIC! this directory and its contents is not removed. Only standard files like the code samples are changed after reinstalling.

File.root <full_path_svar>{, <dirType_sexp>}

Returns the canonical path from the file system root to the default "<pref base drive>/rfo-basic/data", the default data directory, in <full_path_svar>. The <pref_base_drive> is expanded to the full absolute path from the file system root, "/".

The system constants in <dirType_sexp> enables easy access to BASIC! and other folders:

```
_Alarms, _App, _AppPath, _Data, _Database, _Dcim, _Documents, _Downloads,  
_External, _Internal, _InternalOnExternal, _InternalOnSdRemovable*, _Mnt,  
_Movies, _Music, _Notifications, _Pictures, _Podcasts, _ProgramPath, _Ringtones,  
_SdRemovable*, _Source, _SourceSamples, _Storage, _System
```

* Full available with Android 4.4 KitKat (API19) and later. At sooner APIs the interpreter searches for sdcard1, sdcard2, extSdcard and sd-ext.

If an _Internal... folder is not been created, it will create by the first call. The folder _Documents is not created on some devices. With File.mkdir you can do it yourself.

Note: "file://" + <full_path_svar> is required if you want to use it directly in other commands.

Example:

```
FILE.ROOT dataPath$, "_Documents"  
fn$ = "file://" + dataPath$
```

```

FILE.EXISTS ok, fn$
IF ok = 0
  FILE.ROOT newFolder$, "_External"
  newFolder$ = "file://" + newFolder$ + "/Documents"
  FILE.MKDIR newFolder$
ENDIF

```

File.exists <lvar>, <path_svar>

Reports if the <path_svar> directory or file exists. If the directory or file does not exist, the <lvar> will contain zero. If the file or directory does exist, the <lvar> will be returned as non-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/".

It is also possible to use an URI with "file://" as start.

Sometimes you get a file path like "/external/images/media/556".

With "file://" + "/external/images/media/556", File.exists is able to handle it.

In this case <path_svar> returns a new absolute file path.

Before this improvement you could write: File.exists ok, dataPath\$ + "/" + file\$

now you have to write:

```
fn$ = dataPath$ + "/" + file$
```

```
File.exists ok, fn$
```

GrabURL <result_svar>, <url_sexp>{[, <timeout_nexp>], <unicode_flag_lexp>}

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. 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 time-out in milliseconds. This is meaningful only if the URL names a resource on a remote host. If the time-out 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.

Attention: The default setting is the opposite of **GrabFile**!

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.

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.

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 as 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 = "" 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 filepaths.

IF <filename_sexp> = "" or is not defined, the current program itself is restarting again.

Be carefully for not to be 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"
! Run , "1"

! RUN "file://" + aPath$ + "/" + BasName$, "1"
ENDIF
! Run "../source/Files.bas", ""
FILE.ROOT dp$, "_Source"
? dp$

RUN "file://" + dp$ + "/" + "Files.bas"
```

Nicolas's Stuff

Program.Info <nexp>|<nvar>

Returns information about the current program being run 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:

Key	Type	Values	Meaning	Example (from the BASIC Editor)
<u>Path</u>	String	Any string	Full path+name of the program currently being executed. The path is relative to BASIC's "source/" directory.	source/my_program.bas
<u>BasName</u>	String	Any string	Name of the program currently being executed.	my_program.bas
<u>SrcPath</u>	String	Any string	Full path to the app's protected space folder. The path is relative to BASIC's "source/" directory. (add reference to adequate Google Developer's resource)	../../../../data/data/com.rfo.basic
<u>UserApk</u>	Numeric	0 or 1	Returns 1 if the program is currently being run from a user <u>APK</u> . Returns 0 otherwise (program being run from the BASIC Editor)	0
<u>PackageName</u>	String	Any string	Package ID (Name)	com.rfo.basic

Zip Commands now in RFO-Basic 1.91 included

Fixes

RUN "" crashes BASIC! #216

See #218 below

Sometimes SELECT crashes, if too many data in List or Array

The intent transfer memory issue solved

Global Value Backdoor, if interrupt happens

Fixed with Command Groups LOCALS and GLOBALS

BUNDLE.SAVE and BUNDLE.LOAD sometimes a key is missed

Fixed

Other enhancements

Add fullscreen support for HTML5 videos #224

Suggestion borrowed from Nicolas Mougino

Prevent BASIC! halt in case of unhandled Intent #221

Suggestion borrowed from Nicolas Mougino

Prevent user APK crashing b/c of bad/missing permission #220

Suggestion borrowed from Nicolas Mougino

Add support for an APK to register file extension(s) #219

Nicolas Mougino suggested a command **COMMAND\$()**,

but this is also **retData** in

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

Workaround:

```
FN.DEF Command$()
Bundle.in recAction$, retData$, retBundleIndex
FN.RTN retData$
FN.END
ftn$ = Command$()
```

RUN command to restart current program #218

An idea from Nicolas Mougino, but use a little different way.

Command PROGRAM.INFO #217

A Suggestion from Nicolas Mougino, but use a some different values.

See also: **Files.root** and **App.installed**

