De Re BASIC!

Version 1.72

January 11, 2013

Table of Contents

Changes in this Version 19

About the Title, De Re BASIC! 19

Technical Editor 19

BASIC! Tutorial 19

BASIC! Operation 19

- Permissions 19
- Editor 19
  - Editing the program 19
  - Line Continuation 20
  - # - Format Line 20
  - Menus 21
- Run 25
  - Menu 26
- Crashes 26

Command Description Syntax 26

- Upper and Lower Case 26
- <nexp>, <sexp> and <lexp> 26
- <nvar>, <svar> and <lvar> 27
- Array[] and Array$[] 27
- {something} 27
- { A | B |C } 27
- X …, X 27
- {,n …,n} 27
- <statement> 27

Numbers 27

Strings 28

Variables 28

- Variable Names 28
- Variable Types 29
- Scalar and Array Variables 29

Arrays 29
Array Commands 30
Data Structures and Pointers in BASIC! 32
What is a Pointer 32
Lists 33
List Commands 33
Bundles 36
Bundle Commands 36
Stacks 37
Stack Commands 38
Queues 39
Comments 39
! - Single Line Comment 39
!! - Block Comment 39
% - Middle of Line Comment 39
Expressions 39
Numeric <nexp> := {<numeric variable>|<numeric constant} {<noperator> <nexp>|<end of line>} 39
Numeric Operators <noperator> 39
Numeric Expression Examples 40
Pre- and Post-Increment Operators 40
Op Equal Assignment Operations 40
String <sexp> := {<string variable>|<string constant>} { + <sexp> | <end of line>} 40
Logical <lexp> 40
Logical Operators 41
Examples of Logical Expressions 41
Assignment Operations 41
LET 41
OpEqual Assignment Operations 42
Math Functions 42
BOR(<nexp1>, <nexp2>) 42
BAND(<nexp1>, <nexp2>) 42
BXOR(<nexp1>, <nexp2>) 42

ABS(<nexp>) 42
SQR(<nexp>) 43
CBRT(<nexp>) 43
RANDOMIZE(<nexp>) 43
RND() 43
CEIL(<nexp>) 43
FLOOR(<nexp>) 43
MOD(<nexp1>, <nexp2>) 43
ROUND(<nexp>) 43
LOG(<nexp>) 43
LOG10(<nexp>) 43
EXP(<nexp>) 44
POW(<nexp1>, <nexp2>) 44
HYPOT(<nexp_x>, <nexp_y) 44
SIN(<nexp>) 44
COS(<nexp>) 44
TAN(<nexp>) 44
COSH(<nexp>) 44
SINH(<nexp>) 44
ATAN2(<nexp_y>, <nexp_x>) 44
TODEGREES(<nexp>) 44
TORADIANS(<nexp>) 44
ASIN(<nexp>) 44
ACOS(<nexp>) 45
ATAN(<nexp>) 45
VAL(<sexp>) 45
LEN(<sexp>) 45
HEX(<sexp>) 45
OCT(<sexp>) 45
BIN(<sexp>) 45
SHIFT(<value_nexp>, <bits_nexp>) 45
CLOCK() 45

ASCII(<sexp>) 45
UCODE(<sexp>) 46
Is_In(<Search_for_sexp>, <Search_in_sexp>{,<start_nexp>}) 46
Starts_with(<Search_for_sexp>, <Search_in_sexp>{,<start_nexp>}) 46
Ends_with(<Look_for_sexp>, <look_in_sexp>) 46
Gr_collision(<object_1_nvar>, <object_2_nvar>) 46
Background() 46
String Functions 47
getError$() 47
CHR$(<nexp>) 47
LEFT$(<sexp>, <nexp>) 47
MID$(<sexp>, <start_nexp>{, <count_nexp>}) 47
REPLACE$(<target_sexp>, <argument_sexp>, <replace_sexp>) 48
RIGHT$(<sexp>, <nexp>) 48
STR$(<nexp>) 48
LOWER$(<sexp>) 48
UPPER$(<sexp>) 48
VERSION$() 48
HEX$(<nexp>) 48
OCT$(<nexp>) 48
BIN$(<nexp>) 48
FORMAT$(<pattern_sexp>, <nexp>) 48
Leading Sign 48
Floating Field 49
Decimal Point 49
Pattern Character # 49
Pattern Character % 49
Overflow 49
Non pattern characters 49
Output Size 49
Examples: 49
User Defined Functions 49

Commands 50
Fn.def name|name$({nvar}|{svar}|Array[]|Array$[], ... {nvar}|{svar}|Array[]|Array$[]) 50
Fn.rtn <sexp>|<nexp> 51
Fn.end 51
Call <user_defined_function> 51
Program Control Commands 51
If - Then - Else - Elseif - Endif 51
For - To - Step - Next 52
F_n.break 52
While <lexp> - Repeat 52
W_r.break 53
Do – Until <lexp> 53
D_u.break 53
GoSub <label>, Return 53
GoTo <label> 54
Run <filename_sexp> {, <data_sexp>} 54
Switch Commands 54
Nesting Switch Operations 55
Sw.begin <nexp>|<sexp> 55
Sw.case <nexp>|<sexp> 55
Sw.break 55
Sw.default 55
Sw.end 56
OnError: 56
onConsoleTouch: 56
consoleTouch.Resume 56
OnBackKey: 56
Back.resume 57
OnMenuKey: 57
MenuKey.Resume 57
OnKeyPress: 57
Key.Resume 57

End 57
Exit 57
READ – DATA – RESTORE Commands 57
Read.data <number>|<string>{,<number>|<string>…,<number>|<string>} 57
Read.next <svar>|<nvar>{,<svar>|<nvar>…, <svar>|<nvar>} 58
Read.from <nexp> 58
Debug Commands 58
Debug.on 58
Debug.off 58
Debug.echo.on 58
Debug.echo.off 58
Debug.print 59
Debug.dump.scalars 59
Debug.dump.array Array[] 59
Debug.dump.bundle <bundlePtr_nexp> 59
Debug.dump.list <listPtr_nexp> 59
Debug.dump.stack <stackPtr_nexp> 59
Debug.show.scalars 59
Debug.show.array Array[] 59
Debug.show.bundle <bundlePtr_nexp> 60
Debug.show.list <listPtr_nexp> 60
Debug.show.stack <stackPtr_nexp> 60
Debug.watch var, var, …, var 60
Debug.show.watch 60
Debug.show.program 61
Debug.show 61
Console I/O 61
Output Console 61
CLS 61
Print <sexp>|<nexp> {,|;} . . . . <sexp>|<nexp>{,|;} 61
Console.save <filename_sexp> 62
User Input 62

Input <Prompt_sexp>, <nvar>|<svar>{, <Default_sexp>|<Default_nexp>} 62
Inkey$ <svar> 63
Text.input <svar>{, <sexp>} 63
TGet <result_svar>, <prompt_sexp> 63
Kb.toggle 64
Kb.hide 64
Working with Files 64
Paths Explained 64
File.Delete <lvar>, <Path_sexp> 65
File.Dir <Path_sexp>, Array$[] 65
File.Exists <lvar>, <Path_sexp> 65
File.Mkdir <Path_sexp> 65
File.Rename <Old_Path_sexp>, <New_Path_sexp> 66
File.Root <svar> 66
File.Size <size_nvar>, <Path_sexp> 66
Text File I/O 66
Text.open {r|w|a}, <File_table_nvar>, <Path_sexp> 66
Text.close <File_table_nvar> 67
Text.readln <File_table_nvar>, <Line_svar> 67
Text.writeln <File_table_nexp>, <parms same as print> 67
Text.position.get <File_table_nvar>, <position_nvar> 67
Text.position.set <File_table_nvar>, <position_nexp> 68
GrabURL <result_svar>, <url_sexp> 68
GrabFile <result_svar>, <path_sexp> 68
Byte File I/O 68
Byte.open {r|w|a}, <File_table_nvar>, <Path_sexp> 68
Byte.close <File_table_nvar> 69
Byte.read.byte <File_table_nvar>, <byte_nvar> 69
Byte.write.byte <File_table_nvar>, <byte_nexp>|<sexp> 69
Byte.read.buffer <File_table_nvar>, <count_nexp>, <buffer_svar> 69
Byte.write.buffer <File_table_nvar>, <sexp> 69
Byte.position.get <File_table_nvar>, <position_nvar> 69

Byte.position.set <File_table_nvar>, <position_nexp> 70
Byte.copy <File_table_nvar>,<output_file_svar> 70
HTML 70
Introduction 70
Commands 70
Html.open {<Show_status_bar_nexp>} 70
Html.load.url <file_sexp> 71
Html.load.string <html_sexp> 71
Html.post <url _sexp>, <list_nexp> 71
Html.get.datalink <data_svar> 71
Html.go.back 72
Html.go.forward 72
Html.close 72
Html.clear.cache 72
Html.clear.history 72
TCP/IP Sockets 73
TCP/IP Client Socket Commands 73
Socket.client.connect <server_ip_sexp>, <port_nexp> 73
Socket.client.read.line <line_svar> 74
Socket.client.read.ready <nvar> 74
Socket.client.read.file <fw_nexp> 74
Socket.client.write.line <line_sexp> 74
Socket.client.write.bytes <sexp> 74
Socket.client.write.file <fr_nexp> 74
TCP/IP Server Socket Commands 74
Socket.myip <svar> 74
Socket.server.create <port_nexp> 75
Socket.server.connect 75
Socket.server.read.line <svar> 75
Socket.server.read.ready <nvar> 75
Socket.server.write.line <sexp> 75
Socket.server.write.bytes <sexp> 75

Socket.server.write.file <fr_nexp>....75
Socket.server.disconnect....75
Socket.server.close....75
Socket.server.client.ip <nvar>....76
FTP Client....76
Ftp.open <url_sexp>, <port_nexp>, <user_sexp>, <pw_sexp>....76
Ftp.close....76
Ftp.put <source_sexp>, <destination_sexp>....76
Ftp.get <source_sexp>, <destination_sexp>....76
Ftp.dir <list_nvar>....76
Ftp.cd <new_directory_sexp>....77
Ftp.rename <old_filename_sexp>, <new_filename_sexp>....77
Ftp.delete <filename_sexp>....77
Ftp.rmdir <directory_sexp>....77
Ftp.mkdir <directory_sexp>....77
Bluetooth....77
Bt.open {0|1}....78
Bt.close....78
Bt.connect {0|1}....78
Bt.disconnect....78
Bt.reconnect....79
Bt.status <nvar>....79
Bt.write <parms same as print>....79
Bt.read.ready <nvar>....79
OnBtReadReady:....79
Bt.onReadReady.Resume....79
Bt.read.bytes <svar>....79
Bt.device.name <svar>....79
Bt.set.uuid <sexp>....80
Miscellaneous Commands....80
Browse <url_sexp>....80
Swap <nvar_a>|<svar_a>, <nvar_b>, <svar_b>....80

Clipboard 80
Clipboard.get <svar> 80
Clipboard.put <sexp> 80
Echo.on 81
Echo.off 81
Encryption 81
Encrypt <pw_sexp>, <source_sexp>, <encrypted_svar> 81
Decrypt <pw_sexp>, <encrypted_svar>, <decrypted_svar> 81
Text To Speech 81
Tts.init 81
Tts.speak <sexp> 81
Speech To Text (Voice Recognition) 81
Stt.listen 82
Stt.results <string_list_ptr_nvar> 82
Timer Interrupts and Commands 83
Timer.set <interval_nexp> 83
onTimer: 83
Timer.Resume 83
Timer.Clear 83
Sample Code 83
Device <svar> 84
Include FileNamePath 84
Pause <ticks_nexp> 84
Popup <message_sexp>, <x_nexp>, <y_nexp>, <duration_nexp> 84
Select <selection_nvar>, < Array$[]>|<list_nexp>, <message_sexp> {,<press_lvar>} 84
Split <result_Array$[]>, <source_sexp>, <test_sexp> 85
Time Year$, Month$, Day$, Hour$, Minute$, Second$ 85
Tone <frequency_nexp>, <duration_nexp> 85
Vibrate <pattern_Array[]>,<nexp> 86
WakeLock <code_nexp> 86
Http.post <url_sexp>, <list_nexp>, <result_svar> 87
MyPhoneNumber <svar> 87

Phone.call <sexp> 87
Phone.rcv.init 87
Phone.rcv.next <state_nvar>, <number_svar> 87
Sms.send <number_sexp>, <message_sexp> 88
Sms.rcv.init 88
Sms.rcv.next <svar> 88
Email.send <recipient_sxep>, <subject_sexp>, <body_sexp> 88
Headset <state_nvar>, <type_svar>, <mic_nvar> 88
Notify <title_sexp>, <subtitle_sexp>, <alert_sexp>, <wait_lexp> 88
Home 89
OnBackGround: 89
Background.Resume 89
SQLITE 89
Overview 89
SQLITE Commands 90
Sql.open <DB_Pointer_nvar>, <DB_Name_sexp> 90
Sql.close <DB_Pointer_nvar> 90
Sql.new_table <DB_Pointer_nvar>, <DB_Name_sexp>, <Table_Name_sexp>, C1$, C2$, ...,CN$ 90
Sql.drop_table <DB_Pointer_nvar>, <Table_Name_sexp> 91
Sql.insert <DB_Pointer_nvar>, <Table_Name_sexp>, C1$, V1$, C2$, V2$, …, CN$, VN$ 91
Sql.query <Cursor_nvar>, <DB_Pointer_nvar>, <Table_Name_sexp>, Columns$, Where$, Order$ 91
Sql.next <Done_lvar>, <Cursor_nvar>, C1V$, C2V$, ..., CNV$ 92
Sql.delete <DB_Pointer_nvar>, <Table_Name_sexp>, Where$ 92
Sql.update <DB_Pointer_nvar>, <Table_Name_sexp>, C1$, V1$, C2$, V2$,...,CN$, VN$: Where$ ... 92
Sql.exec <DB_Pointer_nvar>, <Command_sexp> 92
Sql.raw_query <Cursor_nvar>, <DB_Pointer_nvar>, <Query_sexp> 92
Graphics 92
Introduction 92
The Graphics Screen and Graphics Mode 92
Display Lists 93
Drawing into Bitmaps 93
Colors 94

Graphics Setup Commands 94
Gr.open alpha, red, green, blue {,<ShowStatusBar_lexp> {, <Orientation_nexp>}} 94
Gr.color alpha, red, green, blue, <style_nexp> 94
Gr.set.AntiAlias <lexp> 95
Gr.set.stroke <nexp> 95
Gr.orientation <nexp> 95
Gr.StatusBar.Show <nexp> 95
Gr.render 95
Gr.screen width, height{, density } 95
Gr.scale x_factor, y_factor 96
Gr.cls 96
Gr.close 96
Gr.front flag 96
Gr.brightness <nexp> 97
Graphical Object Creation Commands 97
Gr.line Object_number, x1, y1, x2, y2 97
Gr.rect Object_number, left, top, right, bottom 97
Gr.oval Object_number, left, top, right, bottom 97
Gr.arc Object_number, left, top, right, bottom, start_angle, sweep_angle, fill_mode 97
Gr.circle Object_number, x, y, radius 98
Gr.set.pixels Object_number, Pixels[] {,x,y} 98
Gr.poly Object_number, List_pointer {,x, y} 98
Hide and Show Commands 99
Gr.hide Object_number 99
Gr.show Object_number 99
Touch Query Commands 99
Gr.touch Touched, x, y 99
Gr.bounded.touch Touched, left, top, right, bottom 100
Gr.touch2 touched, x, y 100
Gr.bounded.touch2 touched, left, top, right, bottom 100
onGRTouch: 100
Gr.onGRTouch.Resume 100

Text Commands 100
Gr.text.align type 100
Gr.text.size n 100
Gr.text.width <nvar>, <sexp> 101
Gr.get.textbounds <sexp>, left, top, right, bottom 101
Gr.text.typeface type 101
Gr.text.bold <lexp> 101
Gr.text.skew <nexp> 101
Gr.text.underline <lexp> 101
Gr.text.strike <lexp> 101
Gr.text.draw <Object_number_nvar>, <x_nexp>, <y_nexp>, <text_object_sexp> 102
Bitmap Commands 102
Gr.bitmap.create bitmap_ptr, width, height 102
Gr.bitmap.load bitmap_ptr, File_name$ 102
Gr.bitmap.size bitmap_ptr, width, height 102
Gr.bitmap.scale dest_ptr, src_ptr, width, height {, smoothing} 102
Gr.bitmap.delete bitmap_ptr 103
Gr.bitmap.crop <new_bitmap_object_nvar>, <source_bitmap_object_nexp>, <x_nexp>, <y_nexp>, <width_nexp>, <height_nexp> 103
Gr.bitmap.save Object_ptr, "filename"{, <quality_nexp>} 103
Gr.bitmap.draw Object_ptr, bitmap_ptr, x , y 103
Gr.get.bmpixel bitmap_ptr, x, y, alpha, red, green, blue 103
Gr.bitmap.drawinto.start Bitmap_Pointer 104
Gr.bitmap.drawinto.end 104
Rotate Commands 104
Gr.rotate.start angle, x, y{,<obj_nvar>} 104
Gr.rotate.end {<obj_nvar>} 104
Camera Commands 104
Gr.camera.select 1|2 105
Gr.camera.shoot bm_ptr 105
Gr.camera.autoshoot bm_ptr {, flash_ mode} 106
Gr.camera.manualShoot bm_ptr {,flash_ mode} 106

Miscellaneous Commands 106
Gr.screen.to.bitmap bm_ptr 106
Gr.get.pixel x, y, alpha, red, green, blue 106
Gr.save "filename" {,<quality_nexp>} 106
Gr.get.position Object_number, x, y 106
Gr.get.value Object_number, tag$, value$ 107
Gr.get.value Text_object_number, "text", theText$ 107
Gr.modify Object_number, parameter_name$, {value|value$} 107
Gr.paint.get <object_nvar> 108
Gr_collision (<object_1_nvar>, <object_2_navr>) 108
Gr.clip <object_nvar>, <left_nexp>, <top_nexp>, <right_nexp>, <bottom_nexp>{,<RO_nexp>} . 108
Gr.NewDL Array[] 109
Audio Interface 110
Introduction 110
The Audio Interface 110
Audio File Types 110
Commands 110
Audio.load <aft_nvar>, <filename_sexp> 110
Audio.play <aft_nexp> 110
Audio.stop 110
Audio.pause 110
Audio.loop 111
Audio.volume <left_nexp>, <right_nexp> 111
Audio.position.current <nvar> 111
Audio.position.seek <nexp> 111
Audio.length <length_nvar>, <aft_nexp> 111
Audio.release <aft_nexp> 111
Audio.isdone <lvar> 111
Audio.record.start <fn_svar> 112
Audio.record.stop 112
SoundPool 112
Introduction 112

Commands 112
Soundpool.open <MaxStreams_nexp> 112
Soundpool.load <soundID_nvar>, <file_path_sexp> 112
Soundpool.unload <soundID_nexp> 113
Soundpool.play <streamID_nvar>, <soundID_nexp>, <rightVolume_nexp>, <leftVolume_nexp>, <priority_nexp>, <loop_nexp>, <rate_nexp> 113
Soundpool.setvolume <streamID_nexp>, <leftVolume_nexp>, <rightVolume_nexp> 113
Soundpool.setrate <streamID_nexp>, <rate_nexp> 113
Soundpool.setpriority <streamID_nexp>, <priority_nexp> 113
Soundpool.pause <streamID_nexp> 113
Soundpool.resume <streamID_nexp> 113
Soundpool.stop <streamID_nexp> 113
Soundpool.release 114
GPS 114
Commands 114
Gps.open 114
Gps.close 114
Gps.provider <svar> 114
Gps.accuracy <nvar> 114
Gps.latitude <nvar> 114
Gps.longitude <nvar> 114
Gps.altitude <nvar> 114
Gps.bearing <nvar> 114
Gps.speed <nvar> 114
Gps.time <nvar> 115
Sensors 115
Introduction 115
Sensor Commands 115
Sensors.list list$[] 115
Sensors.open t1 {,t2,...,tn} 116
Sensors.read sensor_type, p1, p2, p3 116
Sensors.close 116

Superuser 116
Commands 116
Su.open 116
Su.write <sexp> 116
Su.read.ready <nvar> 116
Su.read.line <svar> 116
Su.close 117
Appendix A - Command List 118
Appendix B – Sample Programs 127
Appendix C – Launcher Shortcut Tutorial 128
Introduction 128
How to Make a Shortcut Application (older versions of Android—prior to Android 4.0) 128
How to Make a Shortcut Application (newer versions of Android—Android 4.0 and later) 129
What you need to know 129
Appendix D – Building a Standalone Application 131
Introduction 131
License Information 131
Before You Start 131
Setting Up the Development Environment 131
Download the BASIC! Source Code 132
Create a New Project in Eclipse 132
Rename the package 133
Modifications to Basic.java 135
Changing the APK Name 136
Testing the APK 136
Installing A BASIC! Program Into the Application 137
Adding Your Image and Audio Files 138
Application ICONs 138
Setting The Version Number and Version Name 139
Permissions 139
Preferences 140
Launch at device boot 142

Finished 142
Appendix E – BASIC! Distribution License 143
Apache Commons 154

Changes in this Version

- Changes in description of continuation character "~"
- Improved Appendix C on Launcher Shortcuts
- Added UCODE command
- Added Density parameter to Gr.screen

About the Title, De Re BASIC!

"De Re" is Latin for "of the thing" or "about".

Technical Editor

The technical editor of this manual is Mike Leavitt of Lansdowne, VA, USA. He is monitoring the user forum at http://rfobasic.freeforums.org/suggestions-for-improving-the-manual-f9.html for corrections and suggestions.

BASIC! Tutorial

A BASIC! user, Nick Antonaccio, has written a very nice tutorial for BASIC! You can find it at http://rfobasic.com.

BASIC! Operation

Permissions

This application requests many permissions, permissions such as sending and receiving SMS messages, making phone calls, record audio, etc. BASIC! does not exercise any of these permissions (except writing to the SDCARD) on its own. These permissions get exercised by the BASIC! programmer, you. You and only you. You exercise these permissions by means of the programs that you write.

If you write a program that uses the "SMS.SEND" command then BASIC! will attempt to send an SMS message. BASIC! must have permission to send SMS messages for this command to work. If you never use the "SMS.SEND" command then BASIC! will never send an SMS message. You are in control.

The source code for BASIC! is available from the BASIC! web site (http://laughton.com/basic/). Please feel free to examine this source code if you have any doubt about the use of these permissions.

Editor

Editing the program

The Editor is where BASIC! programs are written and edited. The operation of the Editor is fairly simple. Tap the screen at the point where you want to edit the program. A cursor will appear. Use the keyboard to edit at the cursor location.

When the Enter key is tapped, the new line will automatically indent to the indent level of the previous line. This feature will not work if the Preference, "Editor AutoIndent," is not checked. This feature also may not work if you are using a software keyboard.

If the program that you are editing has been given a name via Save or Load then that program name will be shown in the title bar.

If your Android device does not have a physical keyboard, you will see a virtual keyboard. If you see the virtual keyboard, then you will see different things depending upon the way you are holding the device. If the device is in landscape mode then you will see a dialog box with a chunk of the program in a small text input area. You can scroll the small chunk of text up and down in this area but you will not be able to see very much of the program at any one time. It is probably best not to try to edit a program in landscape mode; hold your device in portrait mode while editing.

On some devices, if you do a long touch on the screen, a dialog box will appear. You can use the selections in the box for selecting, copying, cutting and pasting of text … among other things. Other devices have different procedures for invoking the cut and paste functions.

Line Continuation

A BASIC! source code line 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 pre-processor 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"
```

- Format Line

If a line has the # character at the end of the line, the key words in that line will be capitalized and the # will be removed.

This feature may not work if you are using a virtual keyboard.

This feature will not work if the Preference, "Editor AutoIndent," is not checked.

Menus

Press the Menu key to access the following menus.

Run

Runs the current program.

If the program has been changed since it was last saved, you will be given an opportunity to save the program before the run is started.

If a run-time error occurs then offending line will be shown as selected in the editor.

Sometimes, if a program is re-run too quickly (less than 10 seconds after the end of the previous run) strange runtime errors may occur. If you are having run-time errors that do not make sense, try waiting a bit longer before re-running

Load

Load is used to load a program file into the editor. Programs must be in the directory, "/sdcard/rfo-basic/source", or one of its subdirectories. Program files must have the extension ".bas"

BASIC! checks to see if the current program in the Editor has been changed when Load is tapped. You will be offered the opportunity to save the program if it has been changed. Load will be restarted after the save is done if you choose to save the program.

The "BASIC! Load File" screen shows a sorted list of .bas files and directories. Directories are denoted by the (d) appended to the directory name. Directory entries are at the top of the list. BASIC! programs will be shown with the .bas extension. If there are files in the /source/ directory (or subdirectories) that do not have the .bas extension, they will not appear in the list.

Tap on a .bas file to load it into the Editor.

Tap on a directory to display the contents of that directory.

Tap on the ".." at the top of list to back up one directory level. The tap will be ignored if the current directory is the /source/ directory.

If you accidentally tap the Load button, you can back out of Load by tapping the BACK key.

Save

Saves the program currently in the editor.

A text input dialog box will appear. Type in the name you want the file saved as and tap OK. The extension .bas will be added to file name if is not already there. If the current program has a name because it was previously loaded or save then that name will be in the text input area.

You can back out of Save by tapping the BACK key.

Clear

The current program in the Editor will be cleared. You will be offered the opportunity to save the current program if it has been changed.

Search

Search for strings in the program being edited. Found strings may be replaced with a different string.

The Search view shows a **Text Window** with the text from the Editor: a **Search For** field and a **Replace With** field.

If there is a block of text currently selected in the Editor, then that text will be placed into the **Search For** field.

The initial location of the search cursor will be at the start of the text regardless of where the cursor was in the Editor text.

Note: The search ignores case. For example, searching for "basic" will find "BASIC" This is because BASIC! converts the whole program to lower case (except characters within quotes) when the program is run.

Next Button

Start the search for the string in the **Search For** field. The search is started at the current cursor location. If the string is found then it will be selected in **Text Window**.

If the **Done Button** is tapped at this point then the Editor will returned to with the found text selected.

If the **Replace Button** is tapped then the selected text will be replaced.

Pressing the **Next Button** again will start a new search starting at the end of the selected or replaced text.

If no matching text is found then a "string not found" message will be shown. If the **Done Button** is tapped the Editor will be returned to with the cursor at the end of the program. Alternatively, you could change the **Search For** text and start a new search.

Replace Button

If Next has found and selected some text then that text will be replaced by the contents of the **Replace With** field.

If no text has been found then the message, "Nothing found to replace" will be shown.

Replace All Button

All occurrences of the **Search For** text are replaced with the **Replace With** text. **Replace All** always starts at the start of the text. The last replaced item will be shown selected in the **Text Window**. The number of items replaced will be shown in a message.

Done Button

Returns to the Editor with the changed text. If there is selected text in the **Text Window** then that text will be shown selected in the Editor.

BACK key

If the BACK key is tapped then the Editor will be returned to with the original text unchanged. All changes made during the Search will be undone. Think of the BACK key as UNDO ALL.

More->Format

The program currently in the Editor will be formatted. The key words will be capitalized. Program lines will be indented as appropriate for the program structure.

More->Delete

The Delete Command is used to delete files and directories. The command should be used for maintaining files and directories that are used in BASIC! but it can also be used to delete any file or directory on the SD Card.

Tapping Delete presents the "BASIC! Delete File" screen. The screen has a sorted list of files and directories. Directories are marked with (d) appended to the name and will appear at the top of the list.

Tapping a file name will cause the "Confirm Delete" dialog box to be shown. Tap the Delete button to delete the file. Tap the No button to dismiss the dialog box and not delete the file.

When a directory name is tapped, the contents of directory are displayed. If the directory is empty the "Confirm Delete" dialog box will be shown. Tap the Delete button to delete the directory. Tap the No button to dismiss to dismiss the dialog box and not delete the directory.

Tapping the ".." at the top of the screen moves up one directory level. Tapping the ".." will have no effect if you are in the root directory.

When BASIC! is first started after installing or re-started after an Exit, the directory listed will be the "/sdcard/rfo-basic" directory. If you have changed directories in previous Delete operations then the directory shown will be last directory that you were in.

Exit Delete by tapping the BACK key.

More -> Preferences

Font Size

Sets the font size (Small, Medium, Large) to be used with the various screens in BASIC!

Screen Colors

Sets the appearance of the Screens. Choose Black text on a White background, White text on a Black background or White text on a Blue background.

Editor Lines

Check the box if the text lines in the Editor should be underlined.

Console Lines

Check the box if the text lines in the output console should be underlined.

Console Typeface

Choose the typeface to be used on the Output Console.

Screen Orientation

Choose to allow the Sensors to determine the orientation of the screens or to set a fixed orientation without regard to the Sensors.

Note: The reverse orientations apply to Android 2.3 or newer.

Editor AutoIndent

Check the box if you want the Editor to do auto indentation. Enabling auto indentation will also enable the formatting of a line that ends with the "#" character.

Some devices are not able to do auto indenting properly. In some of those devices the AutoIndent feature may cause the Editor to be unusable. If that happens, turn off AutoIndent.

Base Drive

Some Android devices have several external storage devices (and some have no physical external storage devices). BASIC! will use the system-suggested device as its base drive. The base drive is the device where the BASIC! "rfo-basic" directory (base directory) is located. The base directory is where BASIC!'s programs and data are stored. If you want to use a different storage device as BASIC!'s base drive, you can change it here.

If your device does have more than one external storage device they will be listed here. Tap the device you want to use as the base drive and press the BACK key. You will then be given the choice of either immediately restarting BASIC! with the new base drive or waiting and doing the restart yourself. If your device has no external storage devices, your one and only choice will "No external storage".

Note: If you have created a Launcher Shortcut (see Appendix C) with files in one base directory but try to execute that shortcut while using a different base directory, the shortcut will fail to execute. You will get an error message.

More -> Commands

The Commands command presents the list of the BASIC! commands and functions as copied from Appendix A of this document.

Tapping an alpha key will cause the command list to scroll to commands that start with that character. There will be no scrolling if there is no command that starts with that character.

Note: You can hide the virtual keyboard with the BACK key. If you do that, you will not be able to get it back until you invoke the Commands function again.

Tapping on a particular command causes that command to be copied to the clipboard (not including the page number) and returning to the Editor. You can then paste the command into your BASIC! program.

More -> About

The About command displays the BASIC! web page for the release of BASIC! that corresponds to the release of the BASIC! that you are using. Make sure that you have a connection to the Internet before selecting About.

More -> Exit

The only way to cleanly exit BASIC! is to use the Exit command.

Pressing the Home key while in BASIC! leaves BASIC! in exactly the same state it was in when the Home key was tapped. If a program was running, it will still be running when BASIC! is re-entered. If you were in the process of deleting, the Delete screen will be shown when BASIC! is re-entered.

Run

Pressing the Menu Run button starts the program running. However, if the source in the Editor has been changed, then the Save dialog will be displayed. You may choose to save the changed source or continue without saving.

The BASIC! Output Console will be presented as soon as the program starts to run. You will not see anything on this screen unless one of the following situations occur:

- the program prints something
- the END statement is executed
- you are in Echo mode
- there is a run-time error.

If the program does not print anything then the only indication you would get that the program has finished is if the program ends with an End statement.

If the program does not contain any executable statements then the message, "Nothing to execute" will be displayed.

Tapping the BACK key will stop a running program. Tapping the BACK key when the program run has ended will restart the Editor.

If the program ended with a run-time error, the line where the error occurred will be shown selected in the Editor. If the error occurred in an INCLUDE file then the INCLUDE statement will be shown selected. While this text is selected the Editor "fling scrolling" will not work. Unselect the text to restore the fling scroller.

The Editor cursor will remain where it was when the Run was started if no run-time error occurred.

Menu

Pressing Menu while a program is running or after the program is stopped will cause the Run Menu to be displayed. (Except when Graphics is running. See the Graphics section for details.)

Stop

If a program is running, the Stop menu item will be enabled. Tapping Stop will stop the running program. Stop will not be enabled if a program is not running.

Editor

Editor will not be enabled if a program is running. If the program has stopped and Editor is thus enabled then selecting Editor will cause the Editor to be re-entered. You could also use the BACK key to do this.

Crashes

BASIC! is a very large and complex program that is constantly under development. From time to time, it will crash. When a crash does happen, you will see a brief message on the screen saying, "BASIC! Crashed. Open status bar to report." You will also see a ticker in the status bar saying, "BASIC! Crashed. Open status bar to report the problem."

When you pull down the status bar, there will be an entry saying, "BASIC! has crashed. Please click here to report the problem." When you click on the item a dialog box will be opened. At this point you can supply the developer with some additional information about what you were doing when the crash occurred. You can also give your email address if you would like the developer to contact you about the problem. You may also tap Cancel and not report the problem.

The report to the developer contains a stack trace and other non-personal information that will help the developer fix the problem. Such reports are vital to making BASIC! a more stable product.

Command Description Syntax

Upper and Lower Case

Commands are described using both upper and lower case for ease of reading. BASIC! converts every character (except those between double quotation marks) to lower case when the program is run.

<nexp>, <sexp> and <lexp>

These notations denote a numeric expression (<nexp>), a string expression (<sexp>), or 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).

<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>.

Array[] and Array$[]

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

{something}

Indicates something optional.

{ A | B |C }

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

```
Text.open {r|w|a}, fn…..
```

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

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

X …, X

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

{,n …,n}

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

<statement>

Indicate an executable BASIC! statement. A <statement> is usually a line of code but may occur within other commands such as: If <lexp> then <statement>.

Numbers

Numbers in BASIC! are double-precision 64-bit IEEE 754 floating point. 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 Format command. For example format("##",99) will print as "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. The Format command 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, you can divide by 100.
- You must type decimal numbers with a leading zero. Using .15 will create a syntax error. Using 0.15 will not generate a syntax error.

Numbers can be converted to strings by using the Format command or the STR$(<nexp>) function. A logical value (false = 0, true <> 0) is a kind of number.

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

Strings

Strings in BASIC! begin and end with quote (") characters. For example: "This is a string" is a string.

Strings can include the quote character by using: \" For example:

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

will print: His name is "Jimbo" Jim Giudice.

New line characters may be inserted into a string using: \n

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

will print:

Jim
Giudice

Other special characters can be inserted into a string using the CHR$() function.

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

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

Variables

Variable Names

A BASIC! Variable is a container for some numeric or string value. Variable names must start with the characters "a" through "z" or "#" or "@". The remaining characters in the variable name may also include the numbers 0 through 9 and the underscore (_).

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" to BASIC!

BASIC! key words should not be used to start the name of a variable. For example, Donut = 5 will be interpreted as Do Nut=5. BASIC! thus will expect this Do statement to be followed by an Until statement somewhere before the program ends. A list of BASIC! commands can be found in Appendix A.

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 all numeric variable names.

"First_Name$", "Street$" and "A$" are all string variable names.

Scalar and Array Variables

There are two classes of variables: Scalars and Arrays. A scalar variable can hold one and only one value. An Array variable can hold many values.

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 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}]

BASIC! arrays can have any number of dimensions of any size. The only limitation is that the total number of elements in any single array must be 10,000 or less.

BASIC! arrays are "ones" 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 tells BASIC! how many indices are going to be used and the sizes of the indices. Some BASIC! Commands

automatically dimension an array. Auto dimensioned array details will be seen in the description for 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.

Array Commands

These commands all operate on Arrays.

Dim Array[<nexp>. . . ,<nexp>]

The DIM command tells BASIC! how many dimensions an array will have and how big those dimensions are. 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]
```

UnDim Array[]

Un-dimensions an array. The command allows the array to be dimensioned again with different dimensions. The command is very useful when in a loop using commands that automatically dimension an array. Array[] is specified without any index. The command is exactly the same as "array.delete"

Array.average <Average_nvar>, Array[]

Finds the average of all the values in numeric array, Array[], and then places the result into <Average_nvar>. Array[] is specified without any index.

Array.copy SourceArray[{<start>{,<length>}], DestinationArray[{{-}<extras>}]

The previously Dimensioned or Loaded SourceArray will be copied to the new DestinationArray.

The copy will begin with the <start> element of the SourceArray if the optional <start> parameter is present. If <start> is 0 or 1 or <start> is not present then the copy will begin with the first element of the SourceArray.

The optional <length> parameter specifies a specific number of elements to copy from the SourceArray. If <length> is not present or if <start>+<length> exceeds the number of elements in the SourceArray then the entire array from <start> to the end of the array will be copied.

The optional <extras> parameter specifies that <extras> empty elements are to be added to the Destination Array before or after the copy. These elements will be added to the start of the array if the optional minus(-) sign is present. If minus is not present then these elements will be added to end of the array.

The arrays may be either numeric or string arrays but they must both be of the same type. The extra elements for a numeric array will be initialized to zero. The extra elements for a string array will be the empty string, "".

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

Array.delete Array[]

Does the same thing as UnDim Array[].

Array.length <Length_nvar>, Array[]

Places the number of elements in Array[] into <Length_nvar>.

Array.load Array[], <nexp>{,<nexp>..,<nexp>}

Loads the Array[] with the list, <nexp>{,<nexp>..,<nexp>}, of values. The Array[] will have a single dimension of the size of the number of values in the list.

The array must not have been previously dimensioned.

The array is specified without an index.

String arrays may be loaded in the same manner except that <sexp> replaces <nexp>.

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 <nexp> parameters, where a comma would normally appear. It may not be used to split a parameter across multiple lines.

Examples are:

```
Array.load  Numbers[], 2, 4, 8 , n^2, 32
Array.load Hours[], 3, 4,7,0, 99, 3, 66~
37, 66, 43, 83~
83, n*5, q/2 +j
Array.load Letters$[], "a", "b","c",d$,"e"
```

Array.max <Max_nvar> Array[]

Finds the maximum of all the values in numeric array, Array[], and then places the result into <Max_nvar>. Array[] is specified without any index.

Array.min <Min_nvar>, Array[]

Finds the minimum of all the values in numeric array, Array[], and then places the result into <Min_nvar>. Array[] is specified without any index.

Array.variance <v_nvar>, Array[]

Finds the variance of all the values in numeric array, Array[], and places the result into <v_nvar>. Array[] is specified without any index.

Array.reverse Array[] | Array$[]

Reverses the order of values in the specified array. Array[] is specified without any index.

Array.shuffle Array[] | Array$[]

Randomly shuffles the values of the specified array. Array[] is specified without any index.

Array.sort Array[] | Array$[]

Sorts the specified array in ascending order. Array[] is specified without any index.

Array.std_dev <sd_nvar>, Array[]

Finds the standard deviation of all the values in numeric array, Array[], and places the result into <sd_nvar>. Array[] is specified without any index.

Array.sum <Sum_nvar>, Array[]

Finds the sum of all the values in numeric array, Array[], and then places the result into the <Sum_nvar> variable. Array[] is specified without any index.

Data Structures and Pointers in BASIC!

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

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.

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 place 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.

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.

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.

List Commands

List.create N|S, <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"

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.

List.add <pointer_nexp>, <nexp>{,<nexp>..,<nexp>}

The values, <nexp>{,<nexp>..,<nexp>} will be added to specified list.

Strings may be added in the same manner except that <sexp> replaces <nexp>.

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 <nexp> parameters, where a comma would normally appear. It may not be used to split a parameter across multiple lines.

Examples are:

```
List.add  Nlist, 2, 4, 8 , n^2, 32

List.add Hours, 3, 4,7,0, 99, 3, 66~
37, 66, 43, 83~
83, n*5, q/2 +j

List.add Name "Bill", "Jones"~
"James", "Barnes"
"Jill", "Hanson"
```

List.add.list <destination_list_pointer_nexp>, <source_list_pointer_nexp>

The elements in the source list will be added to the end of the destination list.

The two lists must be of the same type (string or numeric).

List.add.array <destination_list_pointer_nexp>,Array$[]|Array[]

The elements of the specified Array will be added to the end of the destination list.

The Array type must be the same as the list type.

The Array is specified without an index.

List.replace <pointer_nexp>,<index_nexp>, <sexp>|<nexp>

The List element specified by <index_nexp> in the List pointed to by <pointer_nexp> will be replaced by the string or numeric expression.

The index is ones based. The first element of the list is 1.

The replacement expression type (string or numeric) must match the List creation type.

List.insert <pointer_nexp>,<index_nexp>, <sexp>|<nexp>

The <sexp> or <nexp> value will be inserted into the list pointed to by the List pointer.

The element will inserted at the given index point.

The index is ones based. The first element of the list is 1.

The inserted element expression type must match the type (string or numeric) used in the creation of the List.

List.remove <pointer_nexp>,<index_nexp>

The List element specified by <index_nexp> in the List pointed to by <pointer_nexp> will be removed from the list.

The index is ones based. The first element of the list is 1.

List.get <pointer_nexp>,<index_nexp>, <svar>|<nvar>

The List element specified by <index_nexp> in the List pointed to by <pointer_nexp> will be returned in the specified string or numeric variable.

The index is ones based. The first element of the list is 1.

The return element variable type must match the type (string or numeric) used in the creation of the List.

List.type <pointer_nexp>, <svar>

The type of list pointed to by the List pointer will be returned in the String variable.

The upper case character "S" will be returned if the List is a list of strings.

The upper case character "N" will be returned if the list is a list of numbers.

List.size <pointer_nexp>, <nvar>

The size of the List pointed to by the List pointer will be returned in the numeric variable.

List.clear <pointer_nexp>

The list pointed to by the List pointer will be cleared. The List's size will be set to zero.

List.search <pointer_nexp>, value|value$, <result_nvar>{,<start_nexp>}

The list pointed to by the List pointer will be searched for the specified string or numeric value. The position of the found value in the list will be returned in result numeric variable. If the value is not found in the list the result numeric variable value will be zero.

If the optional start expression parameter is present, the search will start at the specified element. The default value is 1.

List.ToArray <pointer_nexp>, Array$[] | Array[]

The list pointed to by the List pointer will be copied into the new, previously non-dimensioned array. The specified array type (string or numeric) must be the same type as the list.

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.

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. The fact that the bundle is represented by a pointer means that bundles can be placed in lists. 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 Commands

Bundle.create <pointer_nvar>

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

Example:

```
Bundle.create bpter
```

Bundle.put <pointer_nexp>, <key_sexp>, <value_nexp>|<value_sexp>

The value expression will be placed into the specified bundle using the specified key.

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

Example:

```
Bundle.put bptr, "first_name", "frank"
Bundle.put bpter,"age", 44
```

Bundle.get <pointer_nexp>, <key_sexp>, <nvar>|<svar>

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.

Example:

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

Bundle.keys <pointer_nexp>, <list_nvar>

A list of the keys currently in the specified bundle will be placed into a new list whose pointer will be returned in <list_nvar>.

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

Example:

```
bundle.keys bpter, list
list.size list, size
  for i = 1 to size
     list.get list, i, key$
     bundle.type bpter, key$, type$
     if type$ = "S"
           bundle.get bpter, key$, value$
           print key$, value$
       else
           bundle.get bpter, key$, value
           print key$, value
           endif
  next i
```

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.

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.

Example:

```
Bundle.type bpter, "age", type$
Print type$   % will print N
```

Bundle.clear <pointer_nvar>

The bundle pointed to by <pointer_nvar> will be cleared of all tags. It will become an empty bundle.

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.

Stack Commands

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>.

Stack.push <ptr_nexep>, <nexp>|<sexp>

Pushes the <nexp> or <sexp> onto the top of the stack designated by <prt_nexp>.

The type of value expression pushed must match the type of the created stack.

Stack.pop <ptr_nexep>, <nvar>|<svar>

Pops the top-of-the-stack value designated by <ptr_nexep> and places it into the <nvar> or <svar>.

The type of the value variable must match the type of the created stack.

Stack.peek <ptr_nexep>, <nvar>|<svar>

Returns the top-of-stack value of the stack designated by <ptr_nexep> 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.

Stack.type <ptr_nexep>, <svar>

The type (numeric or string) of the stack designated by <ptr_nexep> 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.

Stack.IsEmpty <ptr_nexep>, <nvar>

If the stack designated by <ptr_nexep> is empty the value returned in <nvar> will be 1. If the stack is not empty the value will be 0.

Stack.clear <ptr_nexep>

The stack designated by <ptr_nexep> will be cleared.

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 insure that there is a queued element remaining

Comments

! - Single Line Comment

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

!! - Block Comment

When a line begins with the "!!" characters, all lines that follow are considered comments and are ignored by BASIC! The Block quoted section ends at the next line that starts with "!!"

% - Middle of Line Comment

If the "%" character appears in a line (except within a quoted string) then rest of the line is a comment.

Expressions

Numeric <nexp> := {<numeric variable>|<numeric constant} {<noperator> <nexp>|<end of line>}

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 parenthesis.

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

Numeric Expression Examples

```
a
a*b + 4/d – 2*(d^2)
a + b + d + rnd()
b + ceil(d/25) + 5
```

Pre- and Post-Increment Operators

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

Note: these operations are implemented by a pre-processor that changes the source code before it is run. If you have a syntax error in a line that contains these operators, the code line will look different. You cannot combine any of these operators with each other or with any assignment operator (=, +=, etc.).

Op Equal Assignment Operations

+=	a += 1 is the same as a = a + 1
*=	b *= 5 + 3 is the same as b = b *(5+3)
-=	c -= d is the same as c = c –d
/=	e /= log(37) + 1 is the same as e = e / (log(37) + 1)

Note: these operations are implemented by a pre-processor which changes the source code before it is run. If you have a syntax error in a line that contains these operators, the code line will look different. If you use one of these operators, you can not use any of the Pre- and Post-Increment Operators on the same line.

String <sexp> := {<string variable>|<string constant>} { + <sexp> | <end of line>}

There is only one string operator: +

This operator is called the concatenation operator. It is used to join two strings

```
Print "abc" + "def"
will print: abcdef
```

Logical <lexp>

Logical expressions produce false or true results. False and true are represented in BASIC! by the numeric value of zero and not zero. False = 0. 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.

<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 the unary NOT (!) operator. Not inverts the truth of a logical expression.

Logical Operators

The Logical operators are listed by precedence with the highest precedence first. Precedence may be modified by using parenthesis.

1. Unary Not !
2. Less than "<", Greater than ">", Less than or equal "<=", Greater than or equal ">="
3. Equal "=", Not Equal "<>"
4. And "&", Or "|"

Examples of Logical Expressions

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

Assignment Operations

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

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

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. The one time you might use LET is when you when you want to have a variable name start with a BASIC! key word (which may not appear at the beginning of a new line).

The statement:

Letter$ = "B" will be 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 an IF statement, you should also use the LET command:

```
If 1 < 2 then LET letter$="B"
```

OpEqual Assignment Operations

+=	a += 1 is the same as a = a + 1
*=	b *= 5 + 3 is the same as b = b *(5+3)
-=	c -= d is the same as c = c –d
/=	e /= log(37) + 1 is the same as e = e / (log(37) + 1)

Note: these operations are implemented by a pre-processor which changes the source code before it is run. If you have a syntax error in a line that contains these operators, the code line will look different.

Math Functions

Math functions act like numeric variables in a <nexp> (or <lexp>).

BOR(<nexp1>, <nexp2>)

The logical bitwise value of <nexp1> OR <nexp2> will be returned. Floating point double precision values will be converted to integers before the operation.

```
BOR(1,2) = 3
```

BAND(<nexp1>, <nexp2>)

The logical bitwise value of <nexp1> AND <nexp2> will be returned. Floating point double precision values will be converted to integers before the operation.

```
BAND(3,1) = 1
```

BXOR(<nexp1>, <nexp2>)

The logical bitwise value of <nexp1> XOR <nexp2> will be returned. Floating point double precision values will be converted to integers before the operation.

```
BXOR(7,1) = 6
```

ABS(<nexp>)

Returns the absolute value of <nexp>.

SQR(<nexp>)

Returns the correctly rounded positive square root of <nexp>.

CBRT(<nexp>)

Returns the cube root of <nexp>.

RANDOMIZE(<nexp>)

Creates a pseudorandom number generator for use with the RND() function.

The RANDOMIZE () function always returns zero.

If no RANDOMIZE() has been executed or if RANDOMIZE(0) is executed then the seed will be based upon the time of day in milliseconds since January 1, 1970 00:00:00 UTC.

If the numeric expression <> 0 then the generator will be created using the expression value as the seed. A non-zero seed will always generate the same sequence of pseudorandom numbers.

The random numbers generated will be greater than zero and less than one. (0 <= n < 1).

RND()

Returns a random number generated by the pseudorandom number generator. If a RANDOMIZE () has not been previously executed then a new random generator will be created using "RANDOMIZE (0)".

CEIL(<nexp>)

Rounds up. 3.X becomes 4 and -3.X becomes -3.

FLOOR(<nexp>)

Rounds down. 3.X becomes 3 and -3.X becomes -4.

MOD(<nexp1>, <nexp2>)

Returns the remainder of <nexp1> divided by <nexp2>.

ROUND(<nexp>)

Returns the closest whole number to <nexp>.

LOG(<nexp>)

Returns the natural logarithm (base e) of <nexp>.

LOG10(<nexp>)

Returns the base 10 logarithm of the <nexp>.

EXP(<nexp>)

Returns e raised to the <nexp> power.

POW(<nexp1>, <nexp2>)

Returns <nexp1> raised to the <nexp2> power.

HYPOT(<nexp_x>, <nexp_y)

Returns SQR(x^2+y^2) without intermediate overflow or underflow.

SIN(<nexp>)

Returns the trigonometric sine of angle <nexp>. The units of the angle are radians.

COS(<nexp>)

Returns the trigonometric cosine of angle <nexp>. The units of the angle are radians.

TAN(<nexp>)

Returns the trigonometric tangent of angle <nexp>. The units of the angle are radians.

COSH(<nexp>)

Returns the trigonometric hyperbolic cosine of angle <nexp>. The units of the angle are radians.

SINH(<nexp>)

Returns the trigonometric hyperbolic sine of angle <nexp>. The units of the angle are radians.

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.)

TODEGREES(<nexp>)

Converts <nexp> angle measured in radians to an approximately equivalent angle measured in degrees.

TORADIANS(<nexp>)

Converts <nexp> angle measured in degrees to an approximately equivalent angle measured in radians.

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.

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.

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.

VAL(<sexp>)

Converts the <sexp> into a number.

LEN(<sexp>)

Returns the length of the <sexp>.

HEX(<sexp>)

Converts the string expression representing a hexadecimal number to a decimal number.

OCT(<sexp>)

Converts the string expression representing an octal number to a decimal number.

BIN(<sexp>)

Converts the string expression representing a binary number to a decimal number.

SHIFT(<value_nexp>, <bits_nexp>)

Bit shift the value expression by <bits_nexp>, the specified number of bits. If the bits expression is < 0, the value will be shifted left. If the bits expression > 0, the bits will be shifted right. The right shift will replicate the sign bit.

CLOCK()

Returns the time in milliseconds since the last boot.

ASCII(<sexp>)

Returns the ASCII value of the first character of <sexp>. 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.

UCODE(<sexp>)

Returns the Unicode value of the first character of <sexp>. If <sexp> is an empty string ("") the value returned will be 65536 (one more than the largest 16-bit Unicode value). If the first character of <sexp> is a valid ASCII character, this function returns the same value as ASCII().

Is_In(<Search_for_sexp>, <Search_in_sexp>{,<start_nexp>})

Searches the <Search_in_sexp> string for the <Search_for_sexp> string.

If the optional <start_nexp> string is present then the search will start at that value otherwise the search will start at 1, the first character. <start_nexp> must be >= 1.

If the <Search_for_sexp> string is not in the <Search_in_sexp> string, the value returned will be 0, otherwise the value returned will be a ones-based index into the <Search_in_sexp> string.

Starts_with(<Search_for_sexp>, <Search_in_sexp>{,<start_nexp>})

If the <Search_in_sexp> string starting at <start_nexp> starts with the <Search_for_sexp> string, then the length of the <Search_for_sexp> string will be returned. Otherwise zero will be returned.

If the optional <start_nexp> is present then the search will start at that value. Otherwise the search will start at 1, the first character. <start_nexp> must be >= 1.

Ends_with(<Look_for_sexp>, <look_in_sexp>)

If the <look_in_sexp> string does not end with the <Look_for_sexp> string, then the value returned will be zero. If the expression does end with the specified expression the value returned will be index into the string where the <Look_for_sexp> string starts. The value will always be >= 1.

Gr_collision(<object_1_nvar>, <object_2_nvar>)

The object <nvars> are the objects' table numbers 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 and oval. In the case of a circle, arc and an oval it will be object's rectangular boundary box that will be used for collision testing, not the actual drawn object.

Background()

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

Sometimes a BASIC! programmer wants 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.

The Background() function returns true (1) if the program is running in the background. It returns false (0) if the program is not running in the background.

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
```

String Functions

getError$()

Return the error message that was not printed due to an "onError" intercept.

CHR$(<nexp>)

Returns the character represented by the value of <nexp>.

The character "C" is hexadecimal 43, so Print CHR$(16*4 + 3) prints: C

<nexp> may have values greater than 255 and thus can be used to generate Unicode characters.

LEFT$(<sexp>, <nexp>)

Returns the left-most <nexp> characters of <sexp>.

If <nexp> = 0 then an empty string ("") will be returned.

If <nexp> is greater than the length of the <sexp> then the entire string will be returned.

MID$(<sexp>, <start_nexp>{, <count_nexp>})

Returns <count_nexp> chars in the string <sexp> beginning at <start_nexp>.

<start _nexp> is ones based. A zero value for <start _nexp> will be changed to one.

If <start _nexp> is greater than the length of the <sexp> then an empty string ("") will be returned.

If <count _nexp> is greater than length of the <sexp> then the characters <start _nexp> through the end of the <sexp> will be returned.

< count _nexp> is optional. If the <count _nexp> parameter is not given then the characters from <start_nexp> to the end of the string will be returned.

REPLACE$(<target_sexp>, <argument_sexp>, <replace_sexp>)

Returns <target_sexp> with all instances of <argument_sexp> replaced with <replace_sexp>.

RIGHT$(<sexp>, <nexp>)

Returns the right-most <nexp> characters.

If <nexp> = 0 then an empty string ("") will be returned.

If <nexp> is greater than the length of the <sexp> then the entire string will be returned.

STR$(<nexp>)

Returns the string representation of <nexp>.

LOWER$(<sexp>)

Returns <sexp> in all lower case characters.

UPPER$(<sexp>)

Returns <nexp> in all upper case characters.

VERSION$()

Returns the version number of BASIC! as a string.

HEX$(<nexp>)

Returns a string representing the hexadecimal representation of the numeric expression.

OCT$(<nexp>)

Returns a string representing the octal representation of the numeric expression.

BIN$(<nexp>)

Returns a string representing the binary representation of the numeric expression.

FORMAT$(<pattern_sexp>, <nexp>)

Returns a string with <nexp> formatted by the pattern <pattern_sexp>.

Leading Sign

The first character generated by FORMAT is a negative (-) character for numbers < 0 or a space for numbers >= 0.

Floating Field

If the first character of the pattern is not "#" or "." or "!" then that character (along with the sign) will become a "floating" field. This pattern character is typically a $. If no floating character is provided then a space character will be substituted. The floating field will always be two characters wide.

Decimal Point

The pattern may have one and only one optional decimal character. If the decimal character is not present in the pattern, then no decimal digits will be output.

The number of "#" characters after the pattern decimal point specifies the number of decimal digits that will be output.

Pattern Character

Each "#" character will be replaced by a digit from the number in the output. If there are more "#" characters than digits, then the # will be replaced by the space character.

Pattern Character %

Each "%" character will be replaced by a digit from the number in the output. If there are more "%" characters than digits, then the % will be replaced by the zero ("0") character.

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.

It is best not to mix # and % characters. Doing so can produce unexpected results.

Non pattern characters

If any character in the pattern (other than the first character) is not # or %, then that character will be copied directly into the output. This feature is usually used for commas.

Output Size

The number of characters output is always the number of characters in the pattern plus the two floating characters.

Examples:

Format$("##,###,###", 1234567) will output: 1,234,567
Format$("%%,%%%,%%%.#", 1234567.89) will output 01,234,567.8
Format$("$###,###", 1234) will output $1,234
Format$("$###,###", -1234) will output -$1,234

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 functions should generally be defined at the start of the program and in particular, they should appear before the places where they are called.

Each time a User Function is called from a User 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.

Commands

Fn.def name|name$({nvar}|{svar}|Array[]|Array$[], ... {nvar}|{svar}|Array[]|Array$[])

If the function name ends with the $ character then the function will return a string otherwise it will return a number.

The parameter list can contain as many parameters as needed, or none at all. The parameters may be numeric or string, scalar or array.

The following are all valid:

```
fn.def pi()
fn.def cut$(a$, left, right)
fn.def sum(a, b, b, d, e, f, g, h, i, j)
fn.def sort(v$[], direction)
```

There are two types of parameters: call by reference and call by value. Call by value means that the calling variable value (or expression) will be copied into the called variable. Changes made to the called variable within the function will not affect the value of the calling variable. Call by reference means that the calling variable value will be 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 will be call by reference. If there is no "&" in front of the calling variable name then the variable will be 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 of always call 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]  %will print: 9, 9
```

Functions may call other functions. A function can even recursively call itself.

All variables within an instantiation of a function are private to that instantiation of 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.

Fn.rtn <sexp>|<nexp>

Causes the function to terminate execution. The value in <sexp>|<nexp> will be the return value of the function. The return expression type, string or number, must match the type of the function name. Fn.rtn statements may appear anywhere in the program that they are needed.

Fn.end

This command ends the definition of a user-defined function. Every function definition must end with fn.end. If the fn.end statement is executed a default value will be returned. If the function type is numeric then the value of 0.0 will be returned. If the function is a string then the empty string ("") will be returned.

Call <user_defined_function>

Executes the user defined function. Any value returned by the function will be discarded.

Program Control Commands

If - Then - Else - Elseif - Endif

The IF statements provide for the conditional execution of statements. The forms provided are:

```
IF <condition> {THEN}
   <statement>
   <statement>
…
   <statement>
{ ELSEIF<condition> { THEN }
    <statement>
    <statement>
…
    <statement> }
{ ELSE
  <statement>
  <statement>
…
  <statement> }
ENDIF
-or-
If  <condition> THEN <statement>  {ELSE <statement> }
```

See the Sample Program file, F04_if_else.bas, for working examples of the IF command.

For - To - Step - Next

For <nvar> = <nexp_1> To <nexp_2> {Step <nexp_3>}
    <statement>
    …..
    <statement>
Next {<nvar>}

{Step <nexp_3>} is optional and may be omitted. If omitted then the Step value will be 1. <nvar> will be assigned the value of <nexp_1>. <nvar> will be compared to <nexp_2>.

If <nexp_3> is positive then
    if <nvar> <= <nexp_2> then
        the statements between the For and Next will be executed.

If <nexp_3> is negative then
    if <nvar> >= <nexp_2> then
        the statements between the For and Next will be executed.

When the Next statement is executed, <nvar> will be incremented or decremented by the Step value and the test will be repeated. The <statement>s will be executed as long as the test is true.

" For-Next loops" can be nested to any level. When For-Next loops are nested, any executed Next statement will apply to innermost executing loop. 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.

F_n.break

The For-Next loop will be terminated if this statement is executed within an active For-Next loop. The statement immediately following the Next will be executed.

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.

W_r.break

The While-Repeat loop will be terminated if W_r.break is executed within an active While-Repeat loop. The statement immediately following the Repeat will be executed.

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.

D_u.break

The Do-Until loop will be terminated if this statement is executed within an active Do-Until loop. The statement immediately following the Until will be executed.

GoSub <label>, Return

The next statement that will be executed will be the statement following <label>.

A <label> is a variable at the start of a line that is followed by the colon ":", character. Labels must stand alone on the line. For example:

```
Loop:
      <statement>
      ….
      <statement>
      GoTo Loop
```

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
```

GoTo <label>

The next statement that will be executed will be the statement following <label>.

Extensive use of the GoTo command in your program should be generally avoided because it consumes excessive system resources. It's present in BASIC! for compatibility with old basic dialects.

You should use structured elements like do..until, while...repeat, etc. in conjunction with f_n.break, etc.

This is especially serious when GoTo commands are executed from inside a IF/ELSE/FOR block jumping to code outside of the block. Doing this consumes many system resources.

This practice may lead your program to a run-time error: "Stack overflow. See manual about use of GOTO."

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 "/sdcard/rfo-basic/source/" If the filename is "program.bas" then the file, "program.bas" in "/sdcard/rfo-basic/source/" 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 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.

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
        <statement3>
        …
        <statement4>
        Sw.break
Sw.case 3
        <statement5>
        …
        <statement6>
        Sw.break
Sw.default
        <statement7>
Sw.end
```

Nesting Switch Operations

Switch operation can NOT be nested. Do not program switch operations within other switch operations.

Sw.begin <nexp>|<sexp>

Begins a switch operation.

The <nexp> or <sexp> will be evaluated. The results will then be compared the <nexp> or <sexp>in the sw.case statements.

If <begin_nexp> = <case_nexp> or if <begin_sexp> = <case_sexp> then the statement following the sw.case will be executed.

Sw.case <nexp>|<sexp>

The type of parameter, numeric or string, in sw.case must match the expression type of the sw.begin statement.

If multiple sw.case statements have the same parameter, only the first sw.case with the matching parameter will be executed

Sw.break

Once a matching sw.case has been found then the statements following the sw.case will be executed until a sw.break is encountered.

When the sw.break is encountered then BASIC! looks for the sw.end statement. Execution will then resume with the statement following the sw.end.

If no sw.break is present in a particular sw.case then subsequent sw.cases will be executed until a sw.break is encountered

Sw.default

If no matching sw.case is found then the sw.default, if present, will be executed. The sw.default must be placed after all the sw.case statements

Sw.end

The sw.end terminates a switch operation. Sw.end must eventually follow a sw.begin.

OnError:

If an "OnError:" label is in a BASIC! program then control will pass to the statement following the "OnError:" label whenever a run-time error occurs.

The error message that would have been printed will not be printed. That error message can be retrieved by the getError$() function.

Be careful. An infinite loop will occur if a run-time error occurs within the OnError code.

You should not place an OnError: statement into your program until the program is fully debugged. Premature use of OnError: will make the program difficult to debug.

The OnError: label must stand alone on the line (as with all labels).

onConsoleTouch:

Tapping on any line on the output Console that has text on it will cause control to be transferred to this label. Note that the touch must be any printed line of text. It cannot be an empty area of the screen.

The primary intent of this interrupt is to provide a means for allowing the user to asynchronously interrupt an executing BASIC! program (not in graphics mode). A common reason for such an interrupt would be to have the program request input via an INPUT statement. See the Sample File, f35_bluetooth.bas, for an example of this.

For detecting screen touches while in graphics mode, use onGRtouch:

onConsoleTouch: must stand alone on the line (as with all labels).

consoleTouch.Resume

Resumes execution at the point in the BASIC! program where the touch occurred.

OnBackKey:

Pressing the BACK key normally halts program execution. The OnBackKey: label will intercept the BACK key event and transfer control to the statement following OnBackKey:

If the program is in Graphics mode then the OnBackKey code should terminate the run. If it does not then there will be no stopping the program (other than using a task killer application).

The primary intent of this intercept is to allow the program to save state and status information before terminating.

The onBackKey: label must stand alone on the line (as with all labels).

Back.resume

If a BACK key tap has been trapped by OnBackKey: then the back.resume will cause the program to resume at the point where the BACK key was tapped.

OnMenuKey:

If this label is in the program and the user taps the Menu key, the currently running program will be interrupted. The statements following this label will be executed.

MenuKey.Resume

If a Menu key tap has been trapped by OnMenuKey: then MenuKey.resume will cause the program to resume at the point where the Menu key was tapped.

OnKeyPress:

If this label is in the program and the user taps any key, the currently running program will be interrupted. The statements following this label will be executed.

Key.Resume

If a key tap has been trapped by OnKeyPress: then key.resume will cause the program to resume at the point where the key was tapped.

End

The End statement prints END on the Text Output Screen and stops the execution of the program. End statements may be placed anywhere in the program.

Exit

The Exit statement causes BASIC! to stop running and exit to the Android home screen.

READ – DATA – RESTORE Commands

These commands approximate the READ, DATA and RESTORE command of Dartmouth Basic.

Read.data <number>|<string>{,<number>|<string>…,<number>|<string>}

The data value 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.

Read.next <svar>|<nvar>{,<svar>|<nvar>..., <svar>|<nvar>}

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.

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

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.

Debug Commands

The debug commands help you debug your program. The execution of all the debug commands is controlled by the Debug.on command. All of the debug commands will be 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 only be executed if you have turned debugging on with Debug.on.

Debug.on

Turns on debug mode. All debug commands will be executed when in the debug mode.

Debug.off

Turns off debug mode. All debug commands (except Debug.on) will be ignored.

Debug.echo.on

Turns on Echo mode. In Echo mode each line of the running BASIC! program will be 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 will be turned off by either the Debug.echo.off or the Debug.off commands.

Debug.echo.off

Turns off the Echo mode.

Debug.print

This command is exactly the same as the Print command except that the print will occur only while in the debug mode.

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.

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.

Debug.dump.bundle <bundlePtr_nexp>

Dumps the Bundle pointed to by the Bundle Pointer numeric expression.

Debug.dump.list <listPtr_nexp>

Dumps the List pointed to by the List Pointer numeric expression.

Debug.dump.stack <stackPtr_nexp>

Dumps the Stack pointed to by the Stack Pointer numeric expression.

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.

There are two buttons in the dialog:

Resume: Resumes execution.

Step: Executes the next line while continuing to display the dialog box.

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.

There are two buttons in the dialog:

Resume: Resumes execution.

Step: Executes the next line while continuing to display the dialog box.

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.

There are two buttons in the dialog:

Resume: Resumes execution.

Step: Executes the next line while continuing to display the dialog box.

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.

There are two buttons in the dialog:

Resume: Resumes execution.

Step: Executes the next line while continuing to display the dialog box.

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.

There are two buttons in the dialog:

Resume: Resumes execution.

Step: Executes the next line while continuing to display the dialog box.

Debug.watch var, var, ..., 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.

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.

There are two buttons in the dialog:

Resume: Resumes execution.

Step: Executes the next line while continuing to display the dialog box.

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 executed line.

There are two buttons in the dialog:

Resume: Resumes execution.

Step: Executes the next line while continuing to display the dialog box.

Debug.show

Pauses the execution of the program and displays a dialog box. The dialog box will contain the last Debug.show.<command> used or by default Debug.show.program .

There are two buttons in the dialog:

Resume: Resumes execution.

Step: Executes the next line while continuing to display the dialog box.

Console I/O

Output Console

BASIC! has two types of output screens: The Output Console and the Graphics Screen. This section deals with the Output Console. See the section on Graphics for information about the Graphics 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.

CLS

The CLS command clears Output Console screen.

Print <sexp>|<nexp> {,|;} . . . . <sexp>|<nexp>{,|;}

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 not be printed until a Print command without a semicolon at the end is executed.

Examples:

```
Print "New", "Message"
```

Prints: New, Message

```
Print "New";" Message"
```

Prints: New Message

```
Print "New" + " Message"
```

Prints: New Message

```
Print 100-1; " Luftballons"
```

Prints: 99.0 *Luftballons*

```
Print format$("##", 99) ; " Luftballons"
```

Prints: 99 *Luftballons*

```
Print "A";"B";"C";
```

Prints: nothing

```
Print "D";"E";"F"
```

Prints: ABCDEF

Console.save <filename_sexp>

The current contents of the Console is saved to the text file specified by the filename string expression.

User Input

Input <Prompt_sexp>, <nvar>|<svar>{, <Default_sexp>|<Default_nexp>}

Generates an input dialog box.

The <Prompt_sexp> will become the dialog box prompt.

If a <nvar> is specified, the numeric result is placed into <nvar>. When requesting numeric input, the only key taps that will be accepted are 0-9, "+", "-" and ".".

If a <svar> is specified, the string result is placed into <svar>.

If a Default expression is given then the expression value will be placed into the input area of the dialog box. The Default expression type must match the <nvar> or <svar> type.

If the user taps the BACK key when the dialog box is being displayed, unless there is an "OnError:" the user will see the messages:

Input dialog cancelled

Execution halted

If there is a statement beginning with the "OnError:" label, execution will resume at the statement following the "OnError:" statement.

Inkey$ <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.

Text.input <svar>{, <sexp>}

This command is similar to "Input" except that it is used to input and/or edit a large quantity of text. It will open a new window with scroll bars and full text editing capabilities.

If the optional <sexp> is present then that text will loaded into the text input window for editing. If <sexp> is not present then the text.input text area will be empty.

When done editing, tap the Finish button. The edited text will be returned in <svar>.

If the BACK key is tapped then all text editing is discarded. <svar> will be returned with the original <sexp> text.

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$
end
```

TGet <result_svar>, <prompt_sexp>

Simulates a terminal. The current contents of the Output Console will be displayed. 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 will be returned in <result_svar>. The prompt and response will be displayed on the Output Console.

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.

Keys may be read using the Inkey$ command. The command may not work in devices with hard or slide-out keyboards.

The command should be followed by "Pause 1000" to insure that the keyboard visibility has time to change.

Kb.hide

Hides the soft keyboard.

Note: If the soft keyboard is showing in the Editor when the Run is started, the Kb.toggle command hides the keyboard. The following code will insure that the soft keyboard is shown regardless of whether the keyboard is showing in the Editor.

```
PAUSE 100
KB.HIDE
PAUSE 1000
KB.TOGGLE
```

Working with Files

BASIC! can work with files anywhere on the SD Card. BASIC! does not access the device's Internal Memory.

Paths Explained

A path describes where a file or directory is located relative to some file or directory.

The top level directory on the SD Card is: "/sdcard/"
BASIC! program files are in: "/sdcard/rfo-basic/source/"
BASIC! data files are in:"/sdcard/rfo-basic/data/"
BASIC! SQLITE databases are in:"/sdcard/rfo-basic/databases/"

All of the BASIC! file I/O commands assume a certain default path. That default path is "/sdcard/rfo-basic/data" except for SQLITE operations. If you want to work with a file that is not in that directory, you will have to specify a path to the appropriate directory.

The "../" path notation means to back up from the current directory by one level. The default path is "/sdcard/rfo-basic/data/" The path "../source/" tells BASIC! to back to up the "/sdcard/rfo-basic/" directory and look into the "/source/" directory.

If you wanted to work with a file in the root directory of the SD Card, the path from the default path is would be: "../../" This tells BASIC! to back up two levels from "/sdcard/rfo-basic/data" to "/sdcard/"

All of these paths get you to a directory where there is a file that you want to read, write or create. To access that specific file, you need to add the filename to the path.

In order to read the file "names.txt" in "/sdcard/rfo-basic/data/", the path would be "names.txt"

In order to read the program file "sines.bas" in "/sdcard/rfo-basic/source", the path would be "../source/sines.bas"

In order to access the music file "rain.mp3" in "/sdcard/music/", the path would be "../../music/rain.mp3"

File.Delete <lvar>, <Path_sexp>

The file or directory at <Path_sexp> will be deleted, if it exists. If the file or directory did not exist before the Delete, 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 "/sdcard/rfo-basic/data/".

File.Dir <Path_sexp>, Array$[]

This command returns the names of the files in the path specified by <Path_sexp>. The filenames are placed into Array$[]. Directory names in the list will have the characters "(d)" after the file names. The files and directories will be sorted alphabetically with the directories at the top of the list.

The default path is "/sdcard/rfo-basic/data/".

File.Exists <lvar>, <Path_sexp>

This command reports if the <Path_sexp> 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 not zero.

The default path is "/sdcard/rfo-basic/data/".

File.Mkdir <Path_sexp>

Before you can use a directory, the directory must exist. The Mkdir command is used to create directories.

The default path is "/sdcard/rfo-basic/data/".

The path to create a new directory, "homes", in "/sdcard/rfo_basic/data/" would be "/homes/".

The path to create a new directory, "icons", in the root directory of the SD Card would be "../../icons".

File.Rename <Old_Path_sexp>, <New_Path_sexp>

The file or directory at Old_Path will be renamed to New_Path.

The default path is "/sdcard/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:

```
Rename "../../testfile.txt", "/data/testfile1.txt"
```

Will remove the file, testfile.txt, from "/sdcard/", place it into "sdcard/rfo-basic/data/" and also rename it to testfile1.txt.

File.Root <svar>

Returns the canonical path from the file system root to "/sdcard/rfo-basic/data/" in <svar>.

File.Size <size_nvar>, <Path_sexp>

The size, in bytes, of the file at <Path_sexp> will be returned in <size_nvar>. If there is no file at <Path_sexp>, the system will generate a run-time error.

The default path is "/sdcard/rfo-basic/data/".

Text File I/O

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 "/sdcard/rfo-basic/data/"

Text.open {r|w|a}, <File_table_nvar>, <Path_sexp>

The file specified by the <Path_sexp> is opened.

The first parameter describes the file I/O mode for this file.

r = Read
w = Write from the start of the file. Writes over any existing data in the file.
a = Append. Writing starts after the last line in the file.

A file table number is placed into <File_table_nvar>. This value is for use in subsequent text.read, text.write 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. 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.

Text.close <File_table_nvar>

The previously opened file represented by <File_table_nvar> will be closed.

Note: It is essential to close an output file if you have written over 8k bytes to it. If you do not close the file then the file will only contain the first 8k bytes.

Text.readln <File_table_nvar>, <Line_svar>

If <File_table_nvar> = -1 then a run-time error will be thrown.

The next line from the specified, previously opened file is read into <Line_svar>.

After the last line in the file has been read, the Characters "EOF" will be placed into <Line_svar>.

This example reads an entire file and prints each line.

```
Text.open r, file_number, "testfile.txt"
Do
Text.read file_number, line$
Print line$
Until line$ = "EOF"
Text.close file_number
```

The file will not 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.

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.

After the last line has been written to the file, the Text.close command should be used to close the file.

Text.position.get <File_table_nvar>, <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.

Note: 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 for append 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.

Text.position.set <File_table_nvar>, <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 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 next read would return EOF. The position returned for Text.position.get at the EOF will be number of lines plus one in the file.

GrabURL <result_svar>, <url_sexp>

The entire source text of the Internet URL <url_sexp> is copied to the <result_svar> string. The Split command can then be used to split the <result_svar> into an array of lines.

GrabFile <result_svar>, <path_sexp>

The entire contents of the file at <path_sexp> will be copied to the <result_svar> string. The Split command could then be used to split the <result_svar> into an array of lines.

The command could be used grab the contents of a text file for direct use with Text.input.

```
GrabFile text$, "MyJournal.txt"
Text.input EditedText$, text$
```

Byte File I/O

Byte file I/O can be used to read and write any type of file (.txt, .jpg, .pdf, .mp3, etc.). The data is read and written one byte at a time.

Byte.open {r|w|a}, <File_table_nvar>, <Path_sexp>

The file specified by the Path string expression is opened. If the path starts with "http…" then an Internet file will be opened.

The default path is "/sdcard/rfo-basic/data/"

The first parameter describes how the file I/O mode for this file.

r = Read
w = Write from the start of the file. Writes over any existing data in the file.
a = Append. Writing starts after the last line in the file.

A file table number is placed into <File_table_nvar>. This value is for use in subsequent Byte.read, Byte.write or Byte.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. 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.

Byte.close <File_table_nvar>

Closes the previously opened file.

Byte.read.byte <File_table_nvar>, <byte_nvar>

If <File_table_nvar> = -1 then a run-time error will be thrown.

A single byte is read from the file and placed into <byte_nvar>. After the last byte in the file has been read the value returned in <byte_nvar> will be -1. Further attempts to read from the file will continue to return the -1 value.

This example reads an entire file and prints each byte.

```
Byte.open r, file_number, "testfile.jpg"
Do
      Byte.read file_number, Byte
      Print Byte
      Until Byte < 0
Byte.close file_number
```

Byte.write.byte <File_table_nvar>, <byte_nexp>|<sexp>

If the second parameter is a numeric expression then the low order 8 bits of the value will be written to the file as a single byte.

If the second parameter is a string expression then the entire string will be written to the file as 8 bit bytes.

Byte.read.buffer <File_table_nvar>, <count_nexp>, <buffer_svar>

Read the specified count of bytes (<count_nexp>) into the buffer string variable (<buffer_svar>) from the file. If the end of file has been reached, the string length (len(<buffer_svar>) will be zero.

Byte.write.buffer <File_table_nvar>, <sexp>

The entire contents of the string expression will be written to the file.

Byte.position.get <File_table_nvar>, <position_nvar>

Get 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.

Byte.position.set <File_table_nvar>, <position_nexp>

Set the position of the next by 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.

Byte.copy <File_table_nvar>,<output_file_svar>

Copies the previously open input file represented by <File_table_nvar> to the file whose path is specified by <output_file_svar>. The default path is "/sdcard/rfo-basic/data/".

If <File_table_nvar> = -1 then a run-time error will be thrown.

The input file will be completely copied to the to the output file. Both files will then be closed.

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.

HTML

Introduction

The BASIC! HTML package is designed to allow the BASIC! programmer to create user interfaces using HTML and JavaScript. The interface provides for interaction between the HTML engine and BASIC!. The HTML programmer can use JavaScript to send messages to BASIC!. 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. The content of all three files are listed in the Appendix B of this document. They are also delivered with the BASIC! apk. 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".html"

Commands

Html.open {<Show_status_bar_nexp>}

This command must be executed before using the HTML interface.

The optional numeric expression requests the status bar to be shown if the value is not 0. The default is to not show the status bar.

Executing a second HTML.OPEN before executing HTML.CLOSE will generate a run-time error.

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 "file:///sdcard/rfo-basic/data/htmlDemo1.html"
```

will load and display the html file residing at the specified location.

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.

Html.load.string <html_sexp>

Loads and displays the HTML contained in the string expression. The base page for this HTML will be:

```
/sdcard/rfo-basic/data/
```

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.

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, and 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 datalinked 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>
```

Html.go.back
Go back one HTML screen, if possible.

Html.go.forward
Go forward one HTML screen, if possible.

Html.close
Closes the HTML engine and display.

Html.clear.cache
Clears the HTML cache.

Html.clear.history
Clears the HTML history.

TCP/IP Sockets

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://automation.whatismyip.com/n09230945.asp"
```

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 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.

TCP/IP Client Socket Commands

Socket.client.connect <server_ip_sexp>, <port_nexp>

Creates a Client TCP/IP socket and attempts to connect to the Server whose IP is specified by the Server IP string expression using the Port specified by Port numeric expression.

This command will not return until the connection has been made or an error is detected. If the device at the specified IP does not respond, the command will time out after a couple of minutes.

Socket.client.read.line <line_svar>

Reads a line from the previously connected Server and places 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.

Socket.client.read.ready <nvar>

If the previously created Client socket has not received a line for reading by socket.client.read.line then <nvar> will be returned as zero. Otherwise a non-zero value will be returned.

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 this command returns a non-zero value.

Socket.client.read.file <fw_nexp>

A file transmitted by a server will be read and written to the file pointed to by <fw_nexp>, which is derived from a previously executed byte.open write command. For example:

```
Byte.open w, fw, "image.jpg"
Socket.client.read.file fw
Byte.close fw
```

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 TCP/IP Server Socket Commands.

Socket.client.write.bytes <sexp>

Send the string expression, <sexp>, to the previously connected Server as UTF-8 characters. No end-of-line characters will be added by BASIC!. If you need a CR or LF character, you will have to make them part of the <sexp>. 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.

Socket.client.write.file <fr_nexp>

A file previously opened for read by byte.open will be transmitted to the client. Example:

```
Byte.open r, fr, "image.jpg"
Socket.client.write.file fr
Byte.close fr
```

TCP/IP Server Socket Commands

Socket.myip <svar>

Returns the IP of the device in <svar>.

If the device is on a WiFi or Ethernet LAN then IP returned will be the device's LAN IP.

Note: This external or WAN IP can be found using:

```
Graburl ip$, "http://automation.whatismyip.com/n09230945.asp"
```

Socket.server.create <port_nexp>

Establishes a Server that will listen to the Port specified by the numeric expression, <port_nexp>.

Socket.server.connect

Directs the previously created Server to accept a connection from the next client in the queue. This command will not return until a connection is made with a Client.

Socket.server.read.line <svar>

Reads a line sent from the previously connected Client and places the line into the line 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.

Socket.server.read.ready <nvar>

If the previously accepted Client socket has not sent a line for reading by socket.server.read.line then <nvar> will be returned as zero. Otherwise a non-zero value will be returned.

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 socket.server.read.ready returns a non-zero value.

Socket.server.write.line <sexp>

Send the string expression, <sexp>, to the previously connected Client as UTF-16 characters. End of line characters will be added to the end of the line.

Socket.server.write.bytes <sexp>

Send the string expression, <sexp>, to the previously connect Client as UTF-8 characters. No end of line characters will be added by BASIC!. If you need a CR of LF character, you will have to make them part of the <sexp>. 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.

Socket.server.write.file <fr_nexp>

A file previously made open for read by byte.open will be transmitted to the client. Example:

```
Byte.open r, fr, "image.jpg"
Socket.server.write.file fr
Byte.close fr
```

Socket.server.disconnect

The connection with the previously connected Client will be closed. A new socket.server.connect can then be executed to connect to the next client in the queue.

Socket.server.close

The previously created Server will be closed. Any currently connected client will be disconnected.

Socket.server.client.ip <nvar>

Returns the IP of the Client currently connected to the Server.

FTP Client

These FTP commands implement a FTP Client

Ftp.open <url_sexp>, <port_nexp>, <user_sexp>, <pw_sexp>

Connects to the specified url and port. Logs onto the server using the specified user name and password. For example:

```
ftp.open "ftp.laughton.com", 21, "basic", "basic"
```

Ftp.close

Disconnects from the FTP server.

Ftp.put <source_sexp>, <destination_sexp>

Uploads specified source file to the specified destination file on connected ftp server.

The source file is relative to the directory, "/sdcard/rfo-basic/data/" If you want to up load 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.

Ftp.get <source_sexp>, <destination_sexp>

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 "/sdcard/rfo-basic/data/" If you want to download a BASIC! source file, the path would be, "../source/xxx.bas".

Ftp.dir <list_nvar>

Creates a list of the files in the current working directory and places that list into a BASIC! List data structure. Directories will have the characters, "(d)" appended to the filename.

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
```

Ftp.cd <new_directory_sexp>

Changes the current working directory to the specified new directory.

Ftp.rename <old_filename_sexp>, <new_filename_sexp>

Renames the specified old filename to the specified new file name.

Ftp.delete <filename_sexp>

Deletes the specified file.

Ftp.rmdir <directory_sexp>

Removes (deletes) the specified directory if and only if that directory is empty.

Ftp.mkdir <directory_sexp>

Creates a new directory of the specified name.

Bluetooth

BASIC! implements Bluetooth 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 BASIC! 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 will wait 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 will be given a list of paired Bluetooth devices and asked to select one to connect to. BASIC! will attempt to connect to that device once it is selected.

You should monitor the state of the Bluetooth using the br.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 insure 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.

Bt.open {0|1}

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.

Bt.close

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

Bt.connect {0|1}

Commands BASIC! to connect 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.

Bt.disconnect

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

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 Listen status).

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

Bt.status <nvar>

Gets the current Bluetooth connection status and place the value in the numeric variable.

0= Nothing going on
1= Listening
2= Connecting
3= Connected

Bt.write <parms same as print>

Write a text line to the Bluetooth connection. The parameters are the same as the PRINT parameters. This command is essentially a PRINT to the Bluetooth connection.

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.

OnBtReadReady:

If a Bluetooth received message is ready (Bt.read.ready would return a non-zero value) the currently running program will be interrupted and execution will resume at the statement after this label. You can then read and handle the message. When done, you can execute the bt.onReadReady.Resume command to resume the interrupted program.

Bt.onReadReady.Resume

Resumes the running of the program at the location where it was interrupted by the Bluetooth Read Ready event.

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 ("").

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.

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.

Some information about 16 bit and 128 bit UUIDs can be found at:

http://farwestab.wordpress.com/2011/02/05/some-tips-on-android-and-bluetooth/

Miscellaneous Commands

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:///sdcard/... " then the file will be open be opened by the application, ThinkFree Mobile. The file types that the free version of ThinkFree Mobile can open are ".txt, .doc,.xls, .rtf".

If your Android device does not already have ThinkFree Mobile Viewer on it, you can find it in the apps section of the Google Play Store. There is also a paid "pro" version that manages additional file types.

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

Swap <nvar_a>|<svar_a>, <nvar_b>, <svar_b>

The values and in "a" and "b" numeric or string variables are swapped.

Clipboard

Clipboard.get <svar>

Copies the current contents of the clipboard into <svar>

Clipboard.put <sexp>

Places <sexp> into the clipboard.

Echo.on

Same as debug.echo.on. See debug.echo.on for details.

Echo.off

Same as debug.echo.off. See debug.echo.off for details.

Encryption

Encrypts and decrypts a string using a supplied password. The encryption algorithm used is "PBEWithMD5AndDES".

Encrypt <pw_sexp>, <source_sexp>, <encrypted_svar>

The <source_sexp> will be encrypted using <pw_sexp>. The encrypted result will be placed into <encrypted_svar>.

Decrypt <pw_sexp>, <encrypted_svar>, <decrypted_svar>

The encrypted data in <encrypted_svar> will be decrypted using <pw_sexp>. The decrypted results will be placed into <decrypted_svar>.

Text To Speech

The speech generated will be spoken in the current default language of the device. The default language is set by:

1. Start the Settings Application
2. Select Voice input & output
3. Select Text-to-speech settings
4. Select Language

Tts.init

This command must be executing before speaking.

Tts.speak <sexp>

Speaks the string expression. The command will not return until the string has been fully spoken.

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.

The Stt.listen commands starts the voice recognition process with a dialog box. The 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.

Stt.listen

Start the voice recognize process by displaying a "Speak Now" dialog box.

Begin speaking.

The recognition will stop when there is a pause in the speaking.

STT.RESULTS should be executed next.

Note: STT.LISTEN is *not* executed to be used in HTML mode.

Stt.results <string_list_ptr_nvar>

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.

Console Mode

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 HTML.CLOSE.

The primary difference is that the STT.LISTEN command is not used in HTML mode. The STT.LISTEN function is done 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 as per above.

The sample file, f37_html_demo.bas, demonstrates the use of voice recognition in HTML mode.

Timer Interrupts and Commands

You can set a timer that will interrupt the execution of your program at some set time interval. When this interrupt occurs program execution will transfer 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 the execution of the program at the point where the timer interrupt occurred.

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 contain an "onTimer" label when this command is executed.

onTimer:

The label after which the timer interrupt code will be placed.

Timer.Resume

Causes program execution to resume at the point where the timer interrupt occurred.

Timer.Clear

Clears the repeating timer. No further timer interrupts will occur.

Sample Code

```
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
```

Device <svar>

Returns information about this Android device in the string variable. The information is the device Model, Device Type and OS.

Include FileNamePath

Inserts another BASIC! program file at this point. The FileNamePath is given without quotes. The command is evaluated before the program is run therefore the path cannot be a string expression.

"Include /functions/DrawGraph.bas" will insert the code from file, "DrawGraph.bas" from the directory, "/sdcard/rfo-basic/source/functions/" into the program

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.

Popup <message_sexp>, <x_nexp>, <y_nexp>, <duration_nexp>

Pops up a small message for a limited duration.

The message is <message_sexp>.

The duration of the message is either 2 seconds or 4 seconds. If <duration_nexp> = 0 the duration will be 2 seconds. If <duration_nexp> <> 0 the duration will be 4 seconds.

The default location for the Popup is the center of the screen. The <x_nexp>, <y_nexp> pair gives a displacement from the center. The values may be negative.

Select <selection_nvar>, < Array$[]>|<list_nexp>, <message_sexp> {,<press_lvar>}

The SELECT command generates a new screen with a list of choices for the user. When the user taps a screen line, the array index for that line will be returned.

<selection_nvar> is the numeric variable into which the selection array index will be returned.

< 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>.

<message_sexp> is a string expression that will be placed into the title bar at the top of the selection screen. It will also be used displayed in a short Popup message unless the message is an empty string ("") in which case there will be no Popup.

The <press_lvar> is optional. If present, the type of user tap—long or short—will be returned in <press_lvar>. The value returned will be 0 (false) if the user selected the item with a short tap. The value returned will be 1 (true) if the user selected the item with a long press.

Split <result_Array$[]>, <source_sexp>, <test_sexp>

The <source_sexp> string will be split into multiple strings that are placed into <result_Array$[]>. The string is split at each location where <test_sexp> occurs. The <test_sexp> occurrences will be removed from the result strings.

The <result_Array$[]> is specified without an index. The array must not have been previously dimensioned.

Example:

```
string$ = "a:b:c:d"
argument$ = ":"
split result$[], string$, argument$

array.length length, result$[]
for i = 1 to length
print  result$[i] + " ";
next i
Print " "
```

Will print: 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

Time Year$, Month$, Day$, Hour$, Minute$, Second$

Returns the current time in the string variables.

Tone <frequency_nexp>, <duration_nexp>

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. If you specify a duration less than this minimum, you will get a run-time error message giving that minimum for that device.

Vibrate <pattern_Array[]>,<nexp>

The vibrate command causes the device to vibrate in the specified, <pattern_Array[]>, pattern.

The pattern is of the form: pause, on,….., pause, on. The pattern may be of any length. There needs to be at least two values to get a single buzz because the first parameter is a pause.

The values for pause and off are durations in milliseconds.

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 or…
If <nexp> > 0 then the pattern play will be cancelled.

See the sample program, f21_sos.bas, for an example of Vibrate.

WakeLock <code_nexp>

The WakeLock command modifies the system screen timeout function. The <code_nexp> may be one of five values. The first four values modify the screen timeout in various ways.

Code	CPU	Screen	Keyboard Light
1	On*	Off	Off
2	On	Dim	Off
3	On	Bright	Off
4	On	Bright	Bright

**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.*

The fifth code value, 5, releases the WakeLock and restores the system screen timeout function.

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 BASIC! be kept running. One way to do this is to put BASIC! into 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 interval expires but the music will continue to play.

Http.post <url_sexp>, <list_nexp>, <result_svar>

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.

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.

Phone.call <sexp>

The phone number contained in the string expression will be called. You device must be connected to a cellular network to make phone calls.

Phone.rcv.init

Prepare to detect phone calls using phone.rcv.next.

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.

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 SMS message.

Sms.rcv.init

Prepare to intercept received SMS using the sms.rcv.next command.

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
```

Email.send <recipient_sxep>, <subject_sexp>, <body_sexp>

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

Headset <state_nvar>, <type_svar>, <mic_nvar>

<state_nvar>: 1.0 if headset plugged in, -1 if no headset plugged in.

<type_svar>: A string describing the device type.

<mic_nvar>: 1.0 if the headset has a microphone, -1 if the headset does not have microphone.

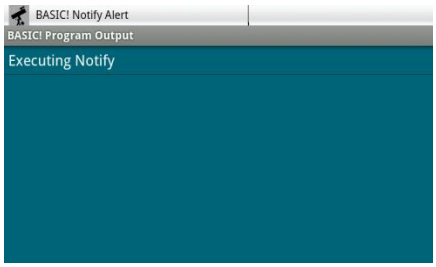
Notify <title_sexp>, <subtitle_sexp>, <alert_sexp>, <wait_lexp>

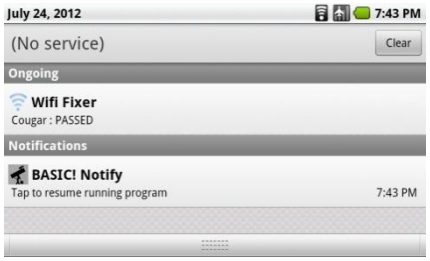
This command will cause a Notify object to be placed in the Notify (Status) bar.

If <wait_lexp> is not zero (true), then the execution of the BASIC! program will be suspended until the user taps the Notify object. If the value is zero (false), the BASIC! program will continue executing.

The Notify object will be removed when the user taps the object.

```
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"
```





Note: the icon that appears in the Notify object will be the icon for the application in user-built apk.

Home

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

OnBackGround:

When the Background/Foreground state changes, if this label is present in the program than the currently executing program will be interrupted and the code following the label will be executed. The Background() function should be used to determine the new state. The interrupted code should be resumed by executing Background.Resume.

Background.Resume

Resumes the execution of the program interrupted by a Background/Foreground state change.

SQLITE

Overview

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 the SQLite Home Page (http://www.sqlite.org/).

An excellent online tutorial on SQL can be found at www.w3schools.com (http://www.w3schools.com/sql/default.asp).

Database files will be created on the SD Card in the directory, "/sdcard/rfo-basic/databases/ ".

SQLITE Commands

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 "data/data/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/table opened at same time. Each opened database/table must have its own, distinct pointer.

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.

Sql.new_table <DB_Pointer_nvar>, <DB_Name_sexp>, <Table_Name_sexp>, C1$, C2$, ...,CN$

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. 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.

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 from the database if the table exists.

Sql.insert <DB_Pointer_nvar>, <Table_Name_sexp>, C1$, V1$, C2$, V2$, ..., CN$, VN$

Inserts a new row of data columns and values into the previously opened database.

<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.)

Sql.query <Cursor_nvar>, <DB_Pointer_nvar>, <Table_Name_sexp>, Columns$, Where$, Order$

Queries the opened database table for some specific data. The command will return a Cursor named <Cursor_nvar> to be used in stepping through query results.

Columns$ is a string 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.

Where$ 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 here. Where$ is an optional parameter. If it is omitted, all rows will be returned.

Order$ 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.

If the Order$ parameter is present, the Where$ parameter must be present. If you want to return all rows, just set Where$ = ""

Sql.next <Done_lvar>, <Cursor_nvar>, C1V$, C2V$, ..., CNV$

Using the Cursor generated by a previous Query command, step to the Next row of data returned by the query.

C1V$, C2V$,CNV$ are the values associated with the columns listed in the Query Columns$ string. If any of your values are numeric values that you need to use in arithmetic operations, you can use the BASIC! VAL(str$) function to convert the value to a number.

The <Done_lvar> parameter is a Boolean used to signal that the last row of the Query returned rows has been read. When the last returned row has been read, <Done_lvar> changes from 0 (false) to 1 (true). When <Done_lvar> becomes true, the Cursor variable is reset to zero. It can no longer be used for sql.next operations. It may be used in a subsequent sql.query statement.

You may have more than one Cursor opened at a time. Each opened Cursor would, of course, have a different variable name.

Sql.delete <DB_Pointer_nvar>, <Table_Name_sexp>, Where$

From a previously opened database table, delete rows selected by the conditions established by Where$. The formation of the Where$ string is exactly the same as described in the sql.query command.

Sql.update <DB_Pointer_nvar>, <Table_Name_sexp>, C1$, V1$, C2$, V2$,...,CN$, VN$: Where$

In a previously opened database table, change column values in specific rows selected by the Where$ parameter. The C$,V$ parameters must be in pairs. The colon character terminates the C$,V$ list and must precede the Where$ in this command.

Sql.exec <DB_Pointer_nvar>, <Command_sexp>

Execute ANY non-query SQL command string ("CREATE TABLE", "DELETE", "INSERT", etc.) using a previously opened database table.

Sql.raw_query <Cursor_nvar>, <DB_Pointer_nvar>, <Query_sexp>

Execute ANY SQL Query command using a previously opened database table and return a Cursor for the results.

Graphics

Introduction

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 graphics 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 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.

The gr.front command can be used to swap the front-most screen between the Output Console and the graphics screen.

When your program ends, the graphics screen will be closed. If you want to keep the graphics screen showing when you program ends, put an infinite loop at the end of the program:

```
! Stay running to keep the
! graphics screen showing
do
until 0
```

Display Lists

BASIC! uses a Display List. The Display List contains pointers to all the graphics objects (circles, lines, etc.) commanded to be drawn. Objects are drawn on the screen in the order in which they appear in the Display List. The Display List objects will not be rendered on the screen until the gr.render command is called.

Each draw object command returns the object's Object Number in something called an Object List. This Object Number can be used to change the object on the fly. You can change any of 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.

Each time an object is added to the Object List, its Object Number is also added to the initial Display List. The user may optionally use the gr.NewDL command to replace the initial display with a custom display list array. This custom display list array will contain some or all of the Object Numbers in the Object List.

The primary 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.

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 Display List. The object number returned by the draw commands while in this mode should be considered invalid and should not be used for any purpose including gr.modify.

The draw-into-bitmap mode is ended when the gr.bitmap.drawinto.end command is issued. 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 bitmap.draw command for that bitmap. The drawn-into bitmap may be drawn at any time before, during or after the draw-into process.

Colors

Android colors consist of a mixture of Red, Green and Blue. Each of the colors can have numerical values ranging from 0 to 255. Black occurs when all three values are zero. White occurs when all three values are 255. Solid Red would occur with Red having a 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 will render the object invisible.

Graphics Setup Commands

Gr.open alpha, red, green, blue {,<ShowStatusBar_lexp> {, <Orientation_nexp>}}

Opens the Graphics Screen and puts BASIC! into the Graphics Mode. The color values become the background color of the graphics screen.

The Status Bar will be shown on the graphics screen if <ShowStatusBar_lexp> is true (not zero).

The orientation upon opening graphics will be determined by the <Orientation_nexp> value. <Orientation_nexp> values are the same as values for the gr.orientation command (see below).

Note: The <ShowStatusBar_lexp> is optional; however, a <ShowStatusBar_lexp> value must be present in order to specify an <Orientation_nexp> value.

The default <Orientation_nexp> value (if <Orientation_nexp> is not specified) is zero (Landscape).

Gr.color alpha, red, green, blue, <style_nexp>

Sets the current color for drawing objects. The current color will be used for whatever graphics objects are subsequently drawn until the next color command is executed.

The BASIC! color command has an additional parameter, <style_nexp>. <Style_nexp> determines the stroking and filling of objects. Stroke parameters are set by commands such as gr.set.stroke (see below) and the various text style commands. The possible values for <style_nexp> are:

> 0: STROKE. Geometry and text drawn with this style will be stroked, 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 both filled and stroked at the same time, respecting the stroke-related fields on the paint.

Gr.set.AntiAlias <lexp>

If <lexp> is false (0), AntiAlias will be turned off. If the value is true (not zero) then AntiAlias will be turned on. 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.

Gr.set.stroke <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.

Gr.orientation <nexp>

Sets the orientation of screen. Landscape is forced when graphics is opened. This can be changed with this command.

-1 = Orientation depends upon the sensors.
0 = Orientation is forced to Landscape.
1 = Orientation is forced to Portrait.

You can monitor changes in orientation by reading the screen width and height using the the gr.screen command.

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.

Gr.render

This command displays all the objects that are in the current working Display List. It is not necessary to have a pause command after a gr.render. The gr.render command will not exit until the contents of the Display List have been fully displayed.

Gr.screen width, height{, density }

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 dpi), and not necessarily the real physical density of the screen.

This command should be executed after the execution of any gr.orientation command. This is because the gr.orientation command swaps the height and width so the values for height and width will be different if the orientation is changed.

Gr.scale x_factor, y_factor

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. For example:

```
! Set the device independent sizes
di_height = 480
di_width = 800

! get the actual width and height

Gr.open 255, 255, 255, 255
Gr.orientation 0
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 upon di_height and di_ width. The drawings will be scaled to fit the device running the program.

Gr.cls

Clears the screen of all objects by disposing of the current Object List and Display List. Creates a new Intial Display List and disposes of any custom display list. All previous gr.color or gr.text {size|align|bold|strike|underline|skew} settings are reset. All previously drawn objects will be deleted. All previous object references are invalidated.

The gr.render command must be called to make the cleared screen visible to the user.

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.

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.

This command is very useful for getting input from the user while in graphics mode. Executing "gr.front 0" will cause the text output screen to be seen by the user. The INPUT command can thus be executed. When the input is received, executing "gr.front 1" will cause the graphics screen to be seen by the user. (Alternatively, the Text.Input command can be used to get text input while in the graphic mode without having to use gr.front.)

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.

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).

Graphical Object Creation Commands

Gr.line Object_number, x1, y1, x2, y2

Creates and inserts a line object into the Display List. The line will start at (x1,y1) and end at (x2,y2). Object_number 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 in 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"

Gr.rect Object_number, left, top, right, bottom

Creates and inserts a rectangle object into the Display List. The rectangle will be located within the bounds of the parameters. The rectangle will or will not be filled depending upon the "gr.color fill" parameter. Object_number 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".

Gr.oval Object_number, left, top, right, bottom

Creates and inserts an oval-shaped object into the Display List. The oval will be located within the bounds of the parameters. The oval will or will not be filled depending upon the gr.color fill parameter. Object_number 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".

Gr.arc Object_number, left, top, right, bottom, start_angle, sweep_angle, fill_mode

Creates and inserts an arc-shaped object into the Display List. The arc will be created within the rectangle described by the parameters. It will start at the specified start_angle and sweep clockwise by the degrees through the specified sweep_angle. If the color fill parameter is true, the fill_mode parameter is activated. If fill_mode is false, the arc will be filled between its end points. If fill_mode is true, the arc will be filled around the center of the arc. A fill_mode of true will produce a wedge or pie shaped object. Object_number 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", "end_angle" and "fill_mode". The value for "fill_mode" is either false (0) or true (not 0).

Gr.circle Object_number, x, y, radius

Creates and inserts a circle object into the Object List. 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 fill parameter. Object_number 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".

Gr.set.pixels Object_number, Pixels[] {,x,y}

Inserts an array of pixel points into the Object List. The Pixels[] array contains pairs of x and y coordinates for each pixel. The Pixels[] array may be any size but should 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 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.

Gr.poly Object_number, List_pointer {,x, y}

Draws a closed polygon of any number of sides.

The List_pointer is a pointer to a List data structure. The list contains x,y coordinate pairs. The first coordinate pair defines the point at which the polygon drawing start. Each subsequent coordinate pair defines a line drawn from the previous coordinate pair to this coordinate pair.

BASIC! will automatically draw the final, polygon-closing line from the last given coordinate pair to the first coordinate pair. This is to insure that the polygon is closed.

The minimum number of coordinate pairs is three pairs (six values). Three pairs define a triangle.

The polygon line width, line color, alpha and fill is determined by previous gr.color and gr.set.stroke commands just like any other drawn object.

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 values for the x,y pair is 0,0. Negative values for the x,y pair are valid.

The gr.modify parameters are "x", "y" and "list".

See the Sample Program file, f30_poly, for working examples of gr.poly.

Hide and Show Commands

Gr.hide Object_number

Hides the object with the specified Object_number. This change will not be visible until the gr.render command is called.

Gr.show Object_number

Shows (unhides) the object with the specified Object_number. This change will not be visible until the gr.render command is called.

Touch Query 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 graphics 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.

Gr.touch Touched, x, y

Tests for a touch on the graphics screen. If the screen was touched, Touched will be returned as true (not 0) with the (x,y) coordinates of the touch. If Touched is false (0), then the screen has not been touched and the (x,y) coordinates are invalid. The command will continue 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 multiply the returned x and y by those same values.

```
gr.touch touched, x, y
Xscaled = x * scale_x
Yscaled = y * scale_y
```

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 multiply left and right by scale_x and multiply top and bottom by scale_y.

Gr.touch2 touched, x, y

The same as gr.touch except that it reports on second simultaneous touch of the screen.

Gr.bounded.touch2 touched, left, top, right, bottom

The same as gr.bounded.touch except that it reports on second simultaneous touch of the screen.

onGRTouch:

If this label is present in the program, the currently running program will be interrupted when the user touches the screen. Execution will then continue at the statement following this label. One of the above gr.touch commands can be executed.

Gr.onGRTouch.Resume

Resumes the execution of the program at the point where the touch interrupt occurred.

Text Commands

Gr.text.align type

Align the text relative to the (x,y) coordinates given in the "gr.text.draw" command.
type values: 1 = Left, 2 = Center, 3 = Right.

Gr.text.size n

Specifies the size of the text in pixels

The size corresponds to the height of the character above the "writing line." Some characters go below the writing line, making the total height of a text field about 30% higher.

Gr.text.width <nvar>, <sexp>

Returns the pixel width of <sexp> in <nvar>. The width is determined by the latest gr.text.size parameter.

Gr.get.textbounds <sexp>, left, top, right, bottom

Gets the boundary rectangle of the string expression <sexp>. The origin of the rectangle is assumed to be 0,0. The value returned for "top" will be a negative number. This is because x coordinate of the Gr.text.draw command specifies the lowest part of the text.

If this is confusing, try running this example:

```
gr.open 255,255,255,255
gr.color 255,255,0,0,0
gr.text.size 40
gr.text.align 1
s$ = "This is a test"
gr.get.textbounds s$,l,t,r,b
print l,t,r,b
gr.rect x,l+10,t+50,r+10,b+50
gr.text.draw x,10,50,s$
gr.render
pause 5000
```

Gr.text.typeface type

Set the text typeface and style.

The values for type are:

1 = Default font
2 = Monospace font
3 = Sans-serif font
4 = Serif font

Gr.text.bold <lexp>

Makes the text appear bold if <lexp> is true (<> 0). 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.

Gr.text.skew <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

Gr.text.underline <lexp>

The drawn text will be underlined if <lexp> is true (<> 0).

Gr.text.strike <lexp>

The text will be drawn with a strike-through line if <lexp> is true (<> 0).

Gr.text.draw <Object_number_nvar>, <x_nexp>, <y_nexp>, <text_object_sexp>

Creates and inserts a text object (<text_object_sexp>) into the Object List. The text object will use the latest color and text preparatory commands. <Object_number_nvar> returns the Object List object number for this text. This object will not be visible until the gr.render command is called.

The y value corresponds to the lowest position of any character in the string.
The 'writing line' lies typically 30% of the gr.text.size higher.

The gr.modify parameters for gr.text.draw are <x_nexp>, <y_nexp>, and <text_object_sexp>. The value for the <text_object_sexp> parameter is a string representing the new text.

Bitmap Commands

Gr.bitmap.create bitmap_ptr, 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 object for use with the other gr.bitmap commands.

Gr.bitmap.load bitmap_ptr, File_name$

Creates a bitmap object from the file specified in the File_name$ string. Returns a pointer to the created bitmap object for use with the gr.bitmap commands.

Bitmap image files are assumed to be located in the "/sdcard/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 if 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.

Gr.bitmap.size bitmap_ptr, width, height

Return the pixel width and height of the bitmap pointed to by bitmap_ptr into the width and height variables.

Gr.bitmap.scale dest_ptr, src_ptr, width, height {, smoothing}

Scales a previously loaded bitmap to the specified width and height and creates a new bitmap. The src_ptr is the bitmap pointer returned by the gr.bitmap.load or other bitmap creating command. The dest_ptr is the bitmap pointer for the new , scaled bitmap.

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.

The optional smoothing logical expression can be used to request that the scaled image not be smoothed. If the expression is zero (false) then the image will not be smoothed. If the optional parameter is not zero (true) or not specified then the image will be smoothed.

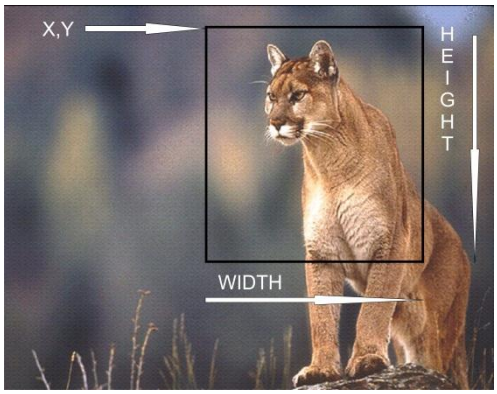
Gr.bitmap.delete bitmap_ptr

Deletes a previously loaded or scaled bitmap. The bitmap's memory is returned to the system.

Gr.bitmap.crop <new_bitmap_object_nvar>, <source_bitmap_object_nexp>, <x_nexp>, <y_nexp>, <width_nexp>, <height_nexp>

The previously loaded source bitmap represented by <source_bitmap_object_nexp> will be cropped. The resulting, new, cropped bitmap object pointer will be returned in <new_bitmap_object_nvar>.

The <x_nexp>, <y_nexp> pair specifies the point with the source bitmap that the crop is to start at. The <width_nexp>, <height_nexp> pair defines the size of the crop.



Gr.bitmap.save Object_ptr, "filename"{, <quality_nexp>}

Saves the specified bitmap to a file. The default path is "/sdcard/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").

Gr.bitmap.draw Object_ptr, bitmap_ptr, x , y

Creates and inserts a bitmap object into the Object List. The bitmap will be drawn with its upper left corner at the provided (x,y) coordinates. The command returns the Display List object number for this bitmap. 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".

Gr.get.bmpixel bitmap_ptr, x, y, alpha, red, green, blue

Returns 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.

Gr.bitmap.drawinto.start Bitmap_Pointer

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 display list. The object number returned by the draw commands while in this mode should be considered 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 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.

Gr.bitmap.drawinto.end

Ends 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 bitmap.draw command for that bitmap.

Rotate Commands

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 around the specified (x,y) point. gr.rotate.start must be eventually followed by gr.rotate.end or you will not get the expected results.

The optional <obj_nvar> will contain the Object list object number of the object to which gr.rotate.start will apply. 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.

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 object to which gr.rotate.end will apply. If you are going to use rotated objects in the array for gr.NewDl then you will need to include the gr.rotate.start object gr.rotate.start and gr.rotate.end objects.

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 "/sdcard/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.

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.

If only one exists 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.

Gr.camera.shoot bm_ptr

The 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 numeric variable. If the camera interface does not, for some reason, take a picture, bm_ptr will be returned with a zero value.

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."

Gr.camera.autoshoot bm_ptr {, flash_ mode}

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
The default, if no parameter is given, is Auto Flash.

The command also stores the captured image into the file, "/sdcard/rfo-basic/data/image.png".

Gr.camera.manualShoot bm_ptr {,flash_ mode}

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.

Miscellaneous Commands

Gr.screen.to.bitmap bm_ptr

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.

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.

Gr.save "filename" {,<quality_nexp>}

Saves the current screen to a file. The default path is "/sdcard/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 file size of the JPEG file is inversely proportional to the quality. Lower quality values produce smaller files.

Gr.get.position Object_number, 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 will be returned in x and top will be returned in y.

Gr.get.value Object_number, tag$, value$

The value represented by tag$ ("left", "radius", etc) in the specified object will be returned in value$.

Gr.get.value Text_object_number, "text", theText$

The text stored in the specified text object will be returned in theText$.

Gr.modify Object_number, parameter_name$, {value|value$}

The object in the display list with the specified Object_number will have its parameter_name$ changed to {value|value$). The parameters for each object are given with descriptions of the commands in this manual.

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, "bitmap", Saturn_ptr`** would change the bitmap of an image of a (preloaded) Galaxy to the image of a (preloaded) Saturn.

The parameters that you supply for a given object are not verified for correctness for the parameter name spelling. The parameter is also not verified for their appropriateness to the specified object. If you are not getting the expected results check the parameter for object appropriateness and spelling. Giving an object an incorrect parameter will not have any effect upon that object.

The results of gr.modify commands will not be observed until a gr.render command is given.

Normally, graphics objects get their alpha channel value (transparency) from the latest gr.color command. This alpha channel value can be explicitly changed by gr.modify without affecting the latest color value. This can be used to make objects slowly appear and disappear.

```
Do
      For a = 1 to 255 step 10
            gr.modify object,"alpha",i
            gr.render
            pause 250
      next a

      For a = 255 to 1 step -10
            gr.modify object,"alpha",i
            gr.render
            pause 250
      next a
until
```

Setting an object's alpha value to 256 tells BASIC! to use the alpha from the latest color value.

Gr.paint.get <object_nvar>

BASIC! has Paint objects. The gr.paint.get command gets the object pointer of the last created paint object. The last created paint object is the paint object associated with a draw object when a draw command is executed. This object 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".

A Paint object includes all paint-related information in addition to the color. This includes font size, style and so forth. Each command that affects the current paint object (gr.color, gr.text.size, etc.) first inherits the current paint and then modifies it to make a new paint object which then becomes the current paint object.

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.

Gr_collision (<object_1_nvar>, <object_2_navr>)

Gr_collision is a function, not a command. The <object nvar>s are the objects' table numbers that were 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 and oval. In the case of a circle, arc and an oval it will be the object's rectangular boundary box that will be used for collision testing, not the actual drawn object.

Gr.clip <object_nvar>, <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 specified "clip rectangle."

The clip rectangle is specified by the "left, top, right, bottom" numeric expressions.

The final, optional, parameter is the Region Operator, <RO_nexp>. The Region Operator prescribes how this clip will interact with the current clip. The full screen is the current clip before the first clip command is issued. The RO values are:

0	**Intersect**
1	**Difference**
2	**Replace**
3	**Reverse Difference**

4	**Union**
5	**XOR**

Examples:

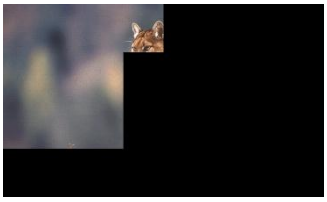
Original

Clip 1

Clip 2

Clip 2 applied to Clip 1 with RO parameter on Clip 2

0 = Intersect

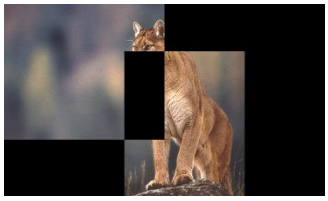


1 = Difference

2 = Replace

3 = Reverse Difference

4 = Union



5 = XOR

gr.clip is a display list object. It can be modified with gr.modify. The modify parameters are "left" "top" "right" "bottom" "RO".

The gr.show and gr.hide commands can be used with the gr.clip object.

Gr.NewDL Array[]

Gr.NewDL replaces the initial display list with a display list composed of an array of object numbers. Zero values on the new display list will be treated as null objects. 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.

Audio Interface

Introduction

The Audio Interface

BASIC! uses 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.

Audio File Types

The Music Player is supposed to be able to play WAV, AAC, MP3, WMA, AMR, OGG and MIDI files. I have tried MP3 and WAV files. Your mileage may vary on these other file types.

Commands

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.

Audio.load <aft_nvar>, <filename_sexp>

Loads a music file into the AFT. The AFT index is returned in <aft_nvar>.

The file must in the "/sdcard/ref-basic/data/" directories or one of its subdirectories.

You can reach outside the "/sdcard/ref-basic/data/" by using path fields in the filename. For example, "../../Music/Blue Danube Waltz.mp3" would access "/sdcard/music/Blue Danube Waltz.mp3"

Audio.play <aft_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 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
```

Audio.stop

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

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.

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.

Audio.volume <left_nexp>, <right_nexp>

Changes the 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 ear perceives a 10db change as twice as loud. A 20db change would be four times as loud.

A 1 db change would be about 0.89. One way to implement a volume control would be set up a volume table with 1db level changes. The following code creates a 16 step table.

```
dim volume[16]
x =1
volume [1] = x
for i = 2 to 16
      x = x * 0.89
      volume [i] = x
next i
```

Your code can select volume values from the table for use in the audio.volume command. The loudest volume would be volume[1].

Audio.position.current <nvar>

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

Audio.position.seek <nexp>

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

Audio.length <length_nvar>, <aft_nexp>

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>.

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.

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
```

Audio.record.start <fn_svar>

Start audio recording using the microphone as the audio source. The recording will be saved to the specified file. The file must have the extension .3GP. Recording will continue until the audio.record.stop command is issued.

Audio.record.stop

Stops the previously started audio recording.

SoundPool

Introduction

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 (SoundPool.pause), individually or as a group, resumed (SoundPool.resume) and stopped (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.

Commands

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.

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.

Soundpool.unload <soundID_nexp>
The specified loaded sound is unloaded.

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.

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.

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.

Soundpool.setpriority <streamID_nexp>, <priority_nexp>
Changes the priority of a playing stream.

The lowest priority is zero.

Soundpool.pause <streamID_nexp>
Pauses the playing of the specified stream. If the stream ID is zero, all streams will be paused.

Soundpool.resume <streamID_nexp>
Resumes the playing of the specified stream. If the stream ID is zero, all streams will be resumed.

Soundpool.stop <streamID_nexp>
Stops the playing of the specified stream.

Soundpool.release
Closes the SoundPool and releases all resources. Soundpool.open can be called to open a new SoundPool.

GPS

These commands provide access to the raw location data provided 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.

Commands

Gps.open
Turns on the GPS hardware and starts it reporting location information. This command must be issued before using any of the other GPS commands.

Gps.close
Turns off 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.

Gps.provider <svar>
Returns the name of the GPS provider in <svar>.

Gps.accuracy <nvar>
Returns the accuracy level in <nvar>.

Gps.latitude <nvar>
Returns the latitude in decimal degrees in <nvar>.

Gps.longitude <nvar>
Returns the longitude in decimal degrees in <nvar>.

Gps.altitude <nvar>
Returns the altitude in meters in <nvar>.

Gps.bearing <nvar>
Returns the bearing in <nvar>.

Gps.speed <nvar>
Returns the speed in meters per second in <nvar>.

Gps.time <nvar>

Returns the returns the UTC time in milliseconds since January 1, 1970.

Sensors

Introduction

Android devices can have several types of Sensors. Currently, Android's pre-defined Sensors are:

Name of Sensor	Type
Accelerometer	1
Magnetic Field	2
Orientation	3
Gyroscope	4
Light	5
Pressure	6
Temperature	7
Proximity	8
Gravity	9
Linear Acceleration	10
Rotation Vector	11

Some details about (most) of these sensors can be found at Android's Sensor Event (http://developer.android.com/reference/android/hardware/SensorEvent.html) web page.

Not all Android devices have all of these Sensors. Some Android devices may have none of these sensors. The BASIC! command, sensors.list, can be used to provide an inventory of the sensors available on a particular device.

Some newer devices may have sensors that are not currently supported by BASIC! Those sensors will be reported as "Unknown, Type = NN" where NN is the sensor type number.

Sensor Commands

Sensors.list list$[]

This command provides a list of the sensors available on a particular Android device. The parameter, list$[], must be an un-dimensioned array. The list will be returned in this array. The elements will contain the names and types of the available sensors. For example, "Gyroscope, Type =4". The following program snip can be used to list the elements of list$[].

```
sensors.list list$[]
array.length size, list$[]
for index = 1 to size
    print list$[ index ]
next index
end
```

The sensors.open command should not be executed before executing this command.

Sensors.open t1 {,t2,...,tn}

Opens a list of sensors for reading. The parameter list is the type numbers of the sensors to be opened. For example, "sensors.open 1, 3", would open the Acceleration and Orientation sensors. 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 drains the battery and increases the background CPU usage.

Note: If your program uses graphics, you should open the sensors before you open graphics.

Sensors.read sensor_type, p1, p2, p3

This command returns that latest values from the sensors specified by the "sensor_type" parameters. The values are returned are placed into the p1, p2 and p3 parameters. The meaning of these parameters depends upon the sensor being read. Not all sensors return all three parameter values. In those cases, the unused parameter values will be set to zero. See Android's Sensor Event web page for the meaning of these parameters.

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 or Menu->Stop.

Superuser

These commands provide for the execution of Superuser commands on rooted devices. See the Sample Program, f36_superuser.bas, for an example using these commands.

Commands

Su.open

Requests Superuser permission.

Su.write <sexp>

Executes a Superuser command.

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.

Su.read.line <svar>

Places the next available response line into the string variable.

Su.close

Exits the Superuser mode.

Appendix A - Command List

! - Single Line Comment, 40
!! - Block Comment, 40
#, 20
- Format Line, 20
% - Middle of Line Comment, 40
ABS(<nexp>), 43
ACOS(<nexp>), 45
Array.average <Average_nvar>, Array[], 30
Array.copy SourceArray[{<start>{,<length>}], DestinationArray[{{-}<extras>}, 30
Array.delete Array[], 31
Array.length <Length_nvar>, Array[], 31
Array.load Array[], <nexp>{,<nexp>..,<nexp>}, 31
Array.max <Max_nvar> Array[], 31
Array.min <Min_nvar>, Array[], 31
Array.reverse Array[] | Array$[], 32
Array.shuffle Array[] | Array$[], 32
Array.sort Array[] | Array$[, 32
Array.std_dev <sd_nvar>, Array[], 32
Array.sum <Sum_nvar>, Array[], 32
Array.variance <v_nvar>, Array[], 31
ASCII(<sexp>), 46
ASIN(<nexp>), 45
ATAN(<nexp>), 45
ATAN2 (<nexp_x>, <nexp_y>), 45
audio.isdone <Boolean_nvar>, 114
audio.length <length_nvar>, <aft_nexp>, 114
audio.load <aft_nvar>, <filename_sexp>, 113
audio.loop, 114
audio.play <aft_nexp>, 113
audio.position.current <nvar>, 114
audio.position.seek <nexp>, 114
audio.record.start <fn_svar>, 115
audio.record.stop, 115
audio.release <aft_nexp>, 114
audio.stop, 113
audio.volume <left_nexp>, <right_nexp>, 114
Back.resume, 57
Background(), 47
Background.Resume, 91
BAND(<nexp1>, <nexp2>), 43
BIN$(<nexp>), 49
BIN(<sexp>), 46
BOR(<nexp1>, <nexp2>), 43
Browse <url_sexp>, 82
Bt.close, 80
Bt.connect {0|1}, 80

Bt.device.name <svar>, 81
Bt.onReadReady.Resume, 81
Bt.open {0|1}, 79
Bt.read.bytes <svar>, 81
Bt.read.ready <navar>, 81
Bt.set.uuid <sexp>, 81
Bt.status <nvar>, 80
Bt.write <parms same a print>, 80
Bundle.clear <pointer_nexp>, 38
Bundle.contain <pointer_nexp>, <key_sexp> , <contains_nvar>, 37
Bundle.create <pointer_nvar>, 36
Bundle.get <pointer_nexp>, <key_sexp>, <nvar>|<svar>, 37
Bundle.keys <pointer_nexp>, <list_nvar>, 37
Bundle.put <pointer_nexp>, <key_sexp>, <value_nexp>|<value_sexp>, 37
Bundle.type <pointer_nexp>, <key_sexp>, <type_svar>, 38
BXOR(<nexp1>, <nexp2>), 43
Byte.close <File_table_nvar>, 70
Byte.copy <File_table_nvar>,<output_file_svar>, 71
Byte.open {r|w|a}, <File_table_nvar>, <Path_sexp>, 69
Byte.position.get <File_table_nvar>, <position_nexp>, 71
Byte.position.set <File_table_nvar>, <position_nexp>, 71
Byte.read.buffer <File_table_nvar>, <count_nexp>, <buffer_svar>, 70
Byte.read.byte <File_table_nvar>, <byte_nvar>, 70
Byte.write.buffer <File_table_nvar>, <sexp>, 71
Byte.write.byte <File_table_nvar>, <byte_nexp>|<sexp>, 70
Call <user_defined_function>, 52
CBRT(<nexp>), 43
CEIL(<nexp>), 44
CHR$ (<nexp>), 47
Clipboard.get <svar>, 82
Clipboard.put <sexp>, 82
Clock(), 46
CLS, 62
Console.save <filename_sexp>, 63
COS(<nexp>), 44
COSH(<nexp>), 45
D_u.break, 54
Debug.dump.array Array[], 59
Debug.dump.bundle <bundlePtr_nexp>, 60
Debug.dump.list <listPtr_nexp>, 60
Debug.dump.scalars, 59
Debug.dump.stack <stackPtr_nexp>, 60
Debug.echo.off, 59
Debug.echo.on, 59
Debug.off, 59
Debug.on, 59
Debug.print, 59
Debug.show, 62

Debug.show.array Array[], 60
Debug.show.bundle <bundlePtr_nexp>, 60
Debug.show.list <listPtr_nexp>, 61
Debug.show.program, 61
Debug.show.scalars, 60
Debug.show.stack <stackPtr_nexp>, 61
Debug.show.watch, 61
Debug.watch var, var, …, var, 61
Decrypt <pw_sexp>, <encrypted_svar>, <decrypted_svar>, 83
Device <svar>, 85
Dim Array[<nexp>. . . ,<nexp>], 30
Do - Until<lexp>, 53
Echo.off, 82
Echo.on, 82
Email.send <recipient_sxep>, <subject_sexp>, <body_sexp>, 90
Encrypt <pw_sexp>, <source_sexp>, <encrypted_svar>, 82
End, 58
Ends_with (<Look_for_sexp>, <look_in_sexp>), 47
Exit, 58
EXP(<nexp>), 44
F_n.break, 53
File.Delete <Boolean_nvar>, <Path_sexp>, 66
File.Dir <Path_sexp>, Array[], 66
File.Exists <Boolean_nvar>, <Path_sexp>, 66
File.Mkdir <Path_sexp>, 66
File.Rename <Old_Path_sexp>, <New_Path_sexp>, 66
File.root <svar>, 67
File.Size <size_nvar>, <Path_sexp>, 67
FLOOR(<nexp>), 44
Fn.def name|name$({nvar}|{svar}|Array[]|Array$[], ….. {nvar}|{svar}|Array[]|Array$[]), 50
Fn.end, 51
Fn.rtn <sexp>|<nexp>, 51
For - To - Step - Next, 52
FORMAT$(<Pattern_sexp>, <nexp>), 49
ftp.cd <new_directory_sexp>, 78
Ftp.Close, 77
ftp.delete <filename_sexp>, 78
ftp.dir <list_navar>, 78
ftp.get <source_sexp>, <destination_sexp>, 78
ftp.mkdir <directory_sexp>, 78
ftp.open <url_sexp>, <port_nexp>, <user_sexp>, <pw_sexp>, 77
ftp.put <source_sexp>, <destination_sexp>, 77
ftp.rename <old_filename_sexp>, <new_filename_sexp>, 78
ftp.rmdir <directory_sexp>, 78
getError$(), 47
GoSub<label>, Return, 54
GoTo <label>, 54
gps.accuracy <nvar>, 117

gps.altitude <nvar>, 118
gps.bearing <nvar>, 118
gps.close, 117
gps.latitude <nvar>, 117
gps.longitude <nvar>, 117
gps.open, 117
gps.provider <svar>, 117
gps.speed <nvar>, 118
gps.time <nvar>, 118
gr.arc Object_number, left, top, right, bottom, start_angle, sweep_angle, fill_mode, 100
Gr.bitmap.create bitmap_ptr, width, height, 104
gr.bitmap.crop <new_bitmap_object_nvar>, <source_bitmap_object_nexp>, <x_nexp>, <y_nexp>,
   <width_nexp>, <height_nexp>, 105
gr.bitmap.delete bitmap_ptr, 105
gr.bitmap.draw Object_ptr, bitmap_ptr, x , w, 106
gr.bitmap.drawinto.end, 106
gr.bitmap.drawinto.start Bitmap_Pointer, 106
gr.bitmap.load bitmap_ptr, File_name$, 104
gr.bitmap.save Object_ptr, “filename”{, <quality_nexp>}, 106
gr.bitmap.scale dest_ptr, src_ptr, Width, Height {, Smoothing}, 105
gr.bitmap.size bitmap_ptr, Width, Height, 105
gr.bounded.touch touched, left, top, right, bottom, 102
gr.bounded.touch2 touched, left, top, right, bottom, 102
gr.brightness <nexp>, 99
gr.camera.autoShoot bm_ptr {,flash_ mode}, 108
Gr.camera.manualShoot bm_ptr {,flash_ mode}, 109
gr.camera.shoot bm_ptr, 108
gr.circle Object_number, x, y, radius, 100
gr.clip <object_nvar>, <left_nexp>,<top_nexp>, <right_nexp>, <bottom_nexp>{,<RO_nexp>}, 111
gr.close, 98
gr.cls, 98
gr.color alpha, red, green, blue, style, 96
gr.front flag, 98
gr.get.bmpixel bitmap_ptr, x, y, alpha, red, green, blue, 106
gr.get.pixel x, y, alpha, red, green, blue, 109
gr.get.position Object_number, x, y, 109
gr.get.textbounds <sexp>, left, top, right, bottom, 103
Gr.get.value Object_number, tag$, value, 109
Gr.get.value Text_Object_number, “text”, theText$, 109
gr.hide Object_number, 101
gr.line Object_number, x1, y1, x2, y2, 99
gr.modify Object_number, parameter_name$, {value|value$}, 109
gr.NewDL Array[], 112
gr.onTouch.Resume, 103
gr.open alpha, red, green, blue {, ShowStatusBar {, Orientation}}, 96
gr.orientation 0|1, 97
gr.oval Object_number, left, top, right, bottom, 99
gr.paint.get <object_nvar>, 110

Gr.poly Object_number, List_pointer {,x,y}, 100
gr.rect Object_number, left, top, right, bottom, 99
gr.render, 97
gr.rotate.end {<obj_nvar>}, 107
gr.rotate.start angle, x, y{,<obj_nvar>}, 107
gr.save “filename” {,<quality_nexp>}, 106, 109
gr.scale x_factor, y_factor, 98
Gr.screen width, height{, density }, 97
gr.screen.to_bitmap bm_ptr, 109
gr.set.AntiAlias <nexp>, 97
gr.set.pixels Object_number, Pixels[] {,x,y}, 100
gr.set.stroke <nexp>, 97
gr.show Object_number, 101
Gr.StatusBar.Show <nexp>, 97
gr.text.align type, 103
gr.text.bold Boolean, 104
gr.text.draw Object_number, x, y, text$, 104
gr.text.size n, 103
gr.text.skew n, 104
gr.text.strike Boolean, 104
gr.text.typeface type, 103
gr.text.underline Boolean, 104
gr.text.width <nvar>, <sexp>, 103
gr.touch touched, x, y, 101
gr.touch2 touched, x, y, 102
gr_collision (<object_1_nvar>, <object_2_navr>), 47, 111
GrabFile <result_svar>,<path_sexp>, 69
GrabURL <result_svar>,<url_sexp>, 69
Graburl ip$, “http://automation.whatismyip.com/n09230945.asp”, 76
Headset <state_nvar>, <type_svar>, <mic_nvar>, 90
HEX$(<nexp>), 49
HEX(<sexp>), 45
Home, 91
Html.clear.cache, 74
HTML.clear.history, 74
Html.close, 74
Html.get.datalink <data_svar>, 73
Html.go.back, 74
Html.go.forward, 74
html.load.string <html_sexp>, 72
Html.load.url <file_sexp>, 72
Html.open {<Show_status_bar_nexp>}, 72
html.post url$, list, 72
http.post url$, list, result$, 89
HYPOT(<nexp_x>, <nexp_y), 44
If - Else -Elseif- Endif, 52
Include FileNamePath, 85
Inkey$ <svar>, 64

Input <Prompt_sexp>, <nvar>|<svar>, {<Default_sexp>|<Default_nexp>}, 63
Is_In(<Search_for_sexp>, <Search_in_sexp>{,<start_nexp>}, 46
Kb.hide, 65
Kb.toggle, 64
Key.Resume, 58
LEFT$ (<sexp>, <nexp>), 48
LEN(<sexp>), 45
List.add <pointer_nexp>, <nexp>{,<nexp>..,<nexp>}, 34
List.add.array <destination_list_pointer_nexp>,Array$[]|Array[], 34
List.add.list <destination_list_pointer_nexp>, <source_list_pointer_nexp>, 34
List.clear <pointer_nexp>, 36
List.create <N|S>, <pointer_nvar>, 34
List.create N|S, <pointer_nvar>, 34
List.get <pointer_nexp>,<index_nexp>, <svar>|<nvar>, 35
List.insert <pointer_nexp>,<index_nexp>, <sexp>|<nexp>, 35
List.remove <pointer_nexp>,<index_nexp>, 35
List.replace <pointer_nexp>,<index_nexp>, <sexp>|<nexp>, 35
List.search <pointer_nexp>, value|value$, <result_nvar>{,<start_nexp>}, 36
List.size <pointer_nexp>, <nvar>, 35
List.ToArray <pointer_nexp>, Array$[] | Array[], 36
List.type <pointer_nexp>, <svar>, 35
LOG(<nexp>), 44
LOG10(<nexp>), 44
LOWER$(<sexp>), 48
MenuKey.Resume, 57
MID$(<sexp>, <start_nexp>, <Count_nexp>}), 48
MOD(<nexp1>, <nexp2>), 44
myPhoneNumber <svar>, 89
Notify <Title_sexp>,< Subtitle_sexp>,<alert_sexp>,<wait_nexp>, 90
OCT$(<nexp>, 49
OCT(<sexp>), 45
OnBackGround:, 91
OnBackKey:, 57
OnBTReadReady:, 81
OnError:, 56
OnKeyPress:, 58
OnMenuKey:, 57
onTimer:, 85
onTouch:, 102
Pause <ticks_nexp>, 86
Phone.call <sexpr>, 89
Phone.rcv.init, 89
Phone.rcv.next <state_nvar>, <number_svar>, 89
Popup <message_sexp>, <x_nexp>, <Y_nexp>, <duration_nexp>, 86
POW(<nexp1>, <nexp2>), 44
Print <sexp>|<nexp> {,|;} . . . . <sexp>|<nexp>{,|;}, 62
RANDOMIZE(<nexp>), 43
Read.data <number>|<string> {,<number>|<string>…,<number>|<string>}, 58

Read.from <nexp>, 59
Read.next <svar>|<nvar>{,<svar>|<nvar>… , <svar>|<nvar>}, 58
REPLACE$(<target_sexp>, <argument_sexp>, <replace_sexp>), 48
RIGHT$(<sexp>, <nexp>), 48
RND(), 44
ROUND(<nexp>), 44
Run <filename_sexp> {, <data_sexp>}, 55
Select <selection_nvar>, < Array$[]>|<list_nexp>, <message_sexp> {,<press_nvar>}, 86
sensors.close, 119
sensors.list list$[], 118
sensors.open t1 {t2,...,tn}, 119
sensors.read sensor_type, p1, p2, p3, 119
SHIFT (<value_nexp>, <bits_nexp>), 46
SIN(<nexp>), 44
SINH(<nexp>), 45
sms.rcv.init, 90
sms.rcv.next <svar>, 90
Sms.send <number_sexp>, <message_sexp>, 89
Socket.client.connect <server_ip_sexp>, <port_nexp>, 75
Socket.client.read.file <fw_nexp>, 75
Socket.client.read.line <line_svar>, 75
Socket.client.read.ready <nvar>, 75
Socket.client.write.bytes <sexp>, 75
Socket.client.write.file <fr_nexp>, 76
Socket.client.write.line <line_sexp>, 75
Socket.myip <svar>, 76
Socket.server.client.ip <nvar>, 77
Socket.server.close, 77
Socket.server.connect, 76
Socket.server.create <port_nexp>, 76
Socket.server.disconnect, 77
Socket.server.read.line <svar>, 76
Socket.server.read.ready <nvar>, 76
Socket.server.write.bytes <sexp>, 77
Socket.server.write.file <fr_nexp>, 77
Socket.server.write.line <sexp>, 76
Soundpool.load <soundID_nvar>, <file_path_sexp>, 115
Soundpool.open <MaxStreams_nexp>, 115
Soundpool.pause <streamID_nexp>, 117
Soundpool.play streamID(nvar), soundID, right_volume, left_volume, priority, loop, rate, 116
Soundpool.release, 117
Soundpool.resume <streamID_nexp>, 117
Soundpool.setpriority <streamID_nexp>, <priority_nexp>, 116
Soundpool.setrate <streamID_nexp>, <rate_nexp>, 116
Soundpool.setvolume <streamID_nexp>, <leftVolume_nexp>, <rightVolume_nexp>, 116
Soundpool.stop <streamID_nexp>, 117
Soundpool.unload <soundID_nexp>, 116
Split <result_Array$[]>, <source_sexp>, <test_sexp>, 87

sql.close DB_Pointer, 92
sql.delete DB_Pointer, Table_Name$, Where$, 94
sql.drop_table DB_Pointer, Table_Name$, 92
sql.exec DB_Pointer, Command$, 94
sql.insert DB_Pointer, Table_Name$, C1$, V1$, C2$, V2$,...,CN$, VN$, 93
sql.new_table DB_Pointer, DB_Name$, Table_Name$, C1$, C2$..,CN$, 92
sql.next Done, Cursor, C1V$, C2V$, .., CNV$, 93
sql.open DB_Pointer, DB_Name$, 92
sql.query Cursor, DB_Pointer, Table_Name$, Columns$, Where$, Order$, 93
sql.raw_query Cursor, DB_Pointer, Query$, 94
sql.update DB_Pointer, Table_Name$, C1$, V1$, C2$, V2$,...,CN$, VN$: Where$, 94
SQR(<nexp>), 43
Stack.clear <ptr_nexep>, 39
Stack.create N|S, <ptr_nvar>, 38
Stack.IsEmpty <ptr_nexep>, <nvar>, 39
Stack.peek <ptr_nexep>, <nvar>|<svar>, 39
Stack.pop <ptr_nexep>, <nvar>|<svar>, 39
Stack.push <ptr_nexep>, <nexp>|<sexp>, 38
Stack.type <ptr_nexep>, <svar>, 39
Starts_with (<Search_for_sexp>, <Search_in_sexp>{,<start_nexp>}, 46
STR$(<sexp>), 48
STT <string_list_ptr_nvar>, 83
STT.LISTEN, 83
Su.close, 120
Su.open, 119
Su.read.line <svar>, 120
Su.read.ready <nvar>, 120
Su.write <sexp>, 119
Sw.begin <nexp>|<sexp>, 56
Sw.break, 56
Sw.case <numeric_constant>|<string_constant>, 56
Sw.default, 56
Sw.end, 56
Swap <nvar_a>|<svar_a>, <nvar_b>, <svar_b>, 82
TAN(<nexp>), 44
Text.close <File_table_nvar>, 67
Text.input <savr>{, <sexp>}, 64
Text.open {r|w|a}, <File_table_nvar>, <Path_sexp>, 67
Text.position.get <File_table_nvar>, <position_nvar>, 68
Text.position.set <File_table_nvar>, <position_nexp>, 69
Text.readln <File_table_nvar>, <Line_svar>, 68
Text.writeln <File_table_nexp>, <parms same as print>, 68
TGet <result_svar>, <prompt_sexp>, 64
Time Year$, Month$, Day$, Hour$, Minute$, Second$, 87
Timer.Clear, 85
Timer.Resume, 85
Timer.set <interval_nexp>, 85
TODEGREES(<nexp>), 45

Tone <frequency_nexp>, <duration_nexp>, 87
TORADIANS(<nexp>), 45
Tts.init, 83
Tts.speak <sexp>, 83
UCODE(<sexp>), 46
UnDim Array[], 30
UPPER$(<sexp>), 49
VAL(<sexp>), 45
VERSION$(), 49
Vibrate <Pattern_Array[]>,<nexp>, 87
W_r.break, 53
WakeLock <code_nexp>, 88
While <lexp> - Repeat, 53

Appendix B – Sample Programs

The programs are loaded into "/sdcard/rfo-basic/source/ Sample Program" when a new release of BASIC! is installed. You can access them by selecting Menu->Load. Tap the "Sample Programs" lines. The sample programs will be listed and can be loaded.

If you load and save one of these programs, the program will be saved in "/sdcard/rfo-basic/source/" not in "/sdcard/rfo-basic/source/ Sample Program"

You can force BASIC! to re-load these programs by:

- Use BASIC! Delete Menu Command, navigate to "sdcard/rfo-basic/source/ Sample Program /"
- Delete the "f01_vxx.xx_read_me file"
- Exit BASIC! using Menu->More->
- Re-enter BASIC!

Appendix C – Launcher Shortcut Tutorial

Introduction

This tutorial will "compile" a BASIC! program and create an "application" that resides on your Android device home page. This "application" will have its own Icon and Name. The official Android name for this type of "application" is "Shortcut." The BASIC! application must be installed for this to work.

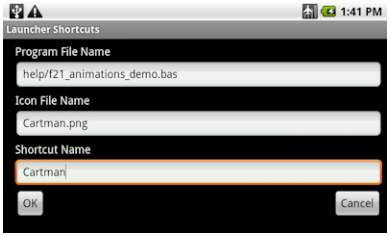
There is also an option to actually build a standalone application .apk file that does not require the BASIC! application to be installed. The process is more difficult but it will result in an application that can be offered on the Android Market. See Appendix D.

How to Make a Shortcut Application (older versions of Android—prior to Android 4.0)

1. Start BASIC!
2. Tap Menu->More->Exit to Exit BASIC!
3. Do a long press on the HOME screen.
4. You should see something like the following:

5. Tap Shortcuts.
6. Scroll down the Select Shortcut page until you see the BASIC! icon with the Launcher Shortcuts Label.

7. Tap the BASIC! Icon.
8. This screen will appear:



9. Fill out the Form exactly as shown.
10. Tap OK.

11. You should see something like this on your HOME screen:
12. Tap the Cartman Shortcut.
13. BASIC! will start and run the Cartman Jumping Demo.

How to Make a Shortcut Application (newer versions of Android—Android 4.0 and later)

1. Select the Apps Page.
2. At the top of the screen, tap Widgets.
3. Scroll horizontally until you see the BASIC! icon that says Launcher Shortcuts.
4. Touch and hold that entry. It will be moved to the Home page.
5. Continue with step 8, above.

What you need to know

- The icon image file must be located in the "/sdcard/rfo-basic/data/" directory.
- The program that you are going to run must be in the "source" directory or one of its sub directories. In this example, the file was located in the SamplesProgram(d) sub-directory of the "source(d)" directory.
- The icon should be a .png file. A Google search for "icon" will reveal thousands for free icons. Just copy your icon into "rfo-basic/data" on the SD card.
- Be very careful to correctly spell the names of the program and icon files. BASIC! does not check to see if these files actually exist during the "compile" process. If you enter the name of an icon file that does not exist, your shortcut will have the generic Android icon. If the file name you specified does not exist, when you tap the Shortcut you will see an error message in the form of program file in the Editor.

- The Shortcut name should be nine (9) characters or less. Android will not show more than nine characters.
- You can create as many shortcuts as you home screen(s) can handle.
- Tapping "Cancel" in the Launcher Shortcuts dialog will simply cancel the operation and return to the home screen.
- If you plan to use a BASIC! Launcher Shortcut, you should always exit BASIC! using Menu->More->Exit. If a Launched program is running, tapping BACK once or twice will exit BASIC back to the Home Screen.

Appendix D – Building a Standalone Application

Note: A BASIC! user, Nicolas Mougin, has created an automated tool for generating standalone applications. This tool can be downloaded from:

http://mougino.free.fr/rfo-basic-app-builder.zip

Using Mr Mougin's tool avoids having to do all of the following.

If you have any questions or problems with this tool you can contact Mr. Mougin and other users of the tool at the BASIC! forum in this thread:

http://rfobasic.freeforums.org/rfo-basic-app-builder-f20.html

Introduction

This document will demonstrate how to create a standalone application from a BASIC! program. The resulting application does not need to have BASIC! installed to run. It will have its own application name and application ICON. It may be distributed in the Android Market or elsewhere. The process involves setting up the Android development environment and making some simple, directed changes to the BASIC! Java source code.

License Information

BASIC! is distributed under the terms of the GNU GENERAL PUBLIC LICENSE. An implication of this is that the source code for your application must be provided to anyone who asks.

Before You Start

Run the Sample Program, f99_my_program.bas.

We are going to turn this program into a practice APK.

Setting Up the Development Environment

1. Download and install the latest version of the Java Development Kit (JDK). Find this by Goggling "java jdk download" and going to the listed oracle.com download site. Do not download from any other site. Note: The JDK download includes the Java Run Time Environment (JRE) which is also needed.
2. Download the Android development SDK installer. Start at: http://developer.android.com/sdk/index.html
3. Continue with: http://developer.android.com/sdk/installing/index.html
   Execute the SDK installer. Install the recommended items.

   (Windows: If you get messages saying that nothing was installed: close the SDK Manager. Go to Start->All Programs->Android SDK Tools. Right click on SDK Manager and select run as administrator.)

If you get a request to start the ADB, do it.

Close the SDK Manager when all packages have been installed.

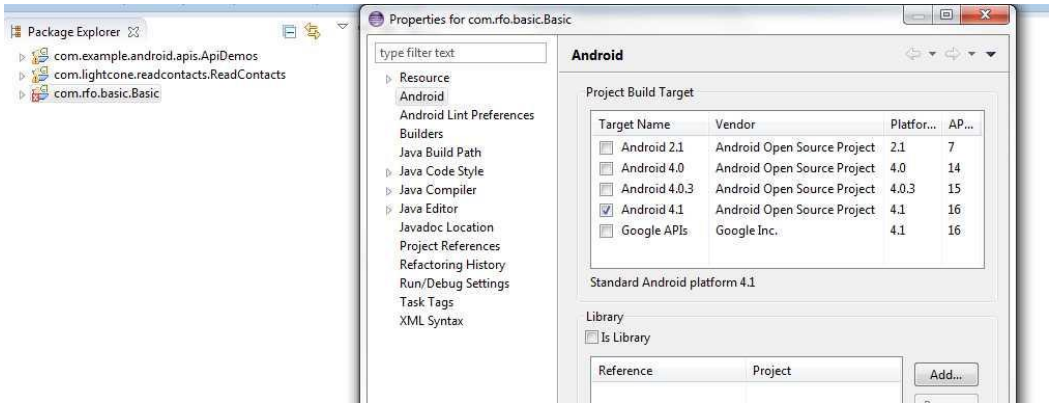
4. Download and install the Eclipse Integrated Development Environment (IDE) from: http://laughton.com/basic/eclipse. Choose the 32 bit or 64 bit version of depending upon your development computer. You can use other versions of Eclipse but these instructions might not work and thus you are on you own.
5. Continue at: http://developer.android.com/sdk/installing/installing-adt.html
6. Start Eclipse. **Carefully** follow the instructions for installing the ADT.
7. Accept any security warnings and allow Eclipse to restart as per instructions.
8. Continue with "Configure the ADT."
9. Download the **latest API** level available no matter what API the level of your device is.
10. Ignore "Updating the ADT" and the rest of the page.
11. Change API level checking from Error to warning:
    a. Select Window->Preferences->Android->List Error Checking
    b. Find "NewApI" and click on it.
    c. In the dropdown list, change the Severity from Error to warning.
    d. Tap Apply

Download the BASIC! Source Code

1. Go to: http://laughton.com/basic/versions/index.html and click on the highest version number.
2. Look for the section heading, "**Download BASIC! Source Code.**" Click on the "**here**" to download the latest "Basic.zip"
3. Unzip this file into a folder named, for example, CatsApp. You should use a different folder for each new APK that you create from Basic.zip.

Create a New Project in Eclipse

1. Select: File -> New -> Project…->Android->Android Project from Existing Code
2. In the Import Projects dialog box, browse to the CatsApp folder(directory) than contains the unzipped Basic folder.
3. Check the project: com.rfo.myapp.Basic. Do not check any other boxes. Click Finish.
4. In the Package Explorer window on the left, right click on com.rfo.basic.Basic.
5. Select Properties (at the bottom of the list).



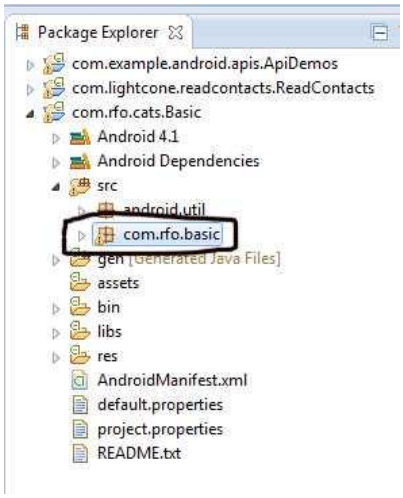
6. Check the highest level of the Android OS available in this list. Do this without regard to the level of OS in your device(s).
7. Click Apply then OK.
8. From the menu, Select: Project->Clean.
9. Check "com.rfo.basic.Basic"
10. Tap Ok.

The Basic source is now ready for making an APK.

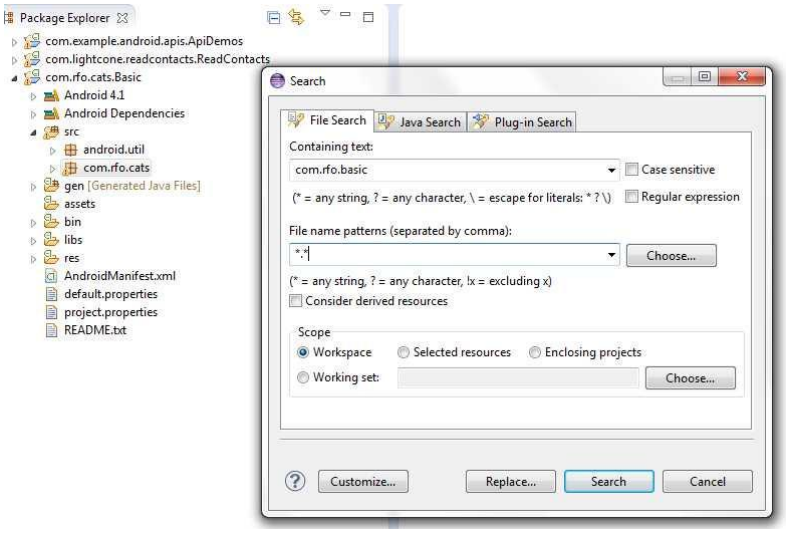
Rename the package

The package name is what makes your application different from every other application that runs on Android devices. No matter what you name your application, it is the package name that Android uses to identify your particular application.

1. In the Package Explorer at the left side of the page tap the name "rfo.com.basic.BASIC".
2. Next Select: File->Rename.
3. Note: In the remainder of this tutorial, we are assuming are application is about cats thus we are using the name "com.rfo.cats" You should, for you own APK, choose a name that matches your application(s).
4. Enter the new name for the package. Let's assume that your application will be named "cats." Change the name of the package to "rfo.com.cats.BASIC"
5. Make sure the Update References check box is checked.
6. Tap OK.
7. In the package explorer, click and open the "src" hierarchy as shown:



8. Select the "com.rfo.basic" as shown.
9. Select File->Rename and rename it to "com.rfo.cats". Make sure the Update References box is checked.
10. From the Menu Bar, select: Search->File.
11. Fill in the dialog like this:

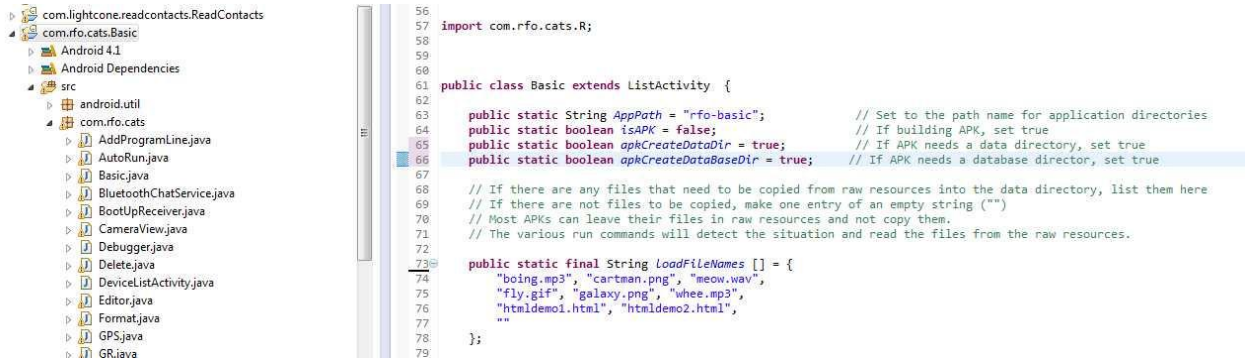


12. Tap Replace.
13. The Replace Text Matches Dialog Box will be shown.
14. Enter "com.rfo.cats" in the "With:" field.
15. Tap Ok
16. A Launch Configuration Change dialog box will be displayed. Tap Yes.
17. Finally, Select: Project -> Clean
18. Tap OK on the Clean Dialog Box.

At this point the package has been successfully renamed. Next we will create a practice APK that you then use to make your own APK.

Modifications to Basic.java

In the Package Explorer, Expand "com.rfo.cats" and then double click on Basic.java. Basic.java will be opened in the window on the right.



Change: AppPath = "rfo-basic" to: AppPath = "rfo-cats"

"rfo-cats" the directory on the SDCARD where your files will be stored, if you choose to have a directory for file for you application. Even if you do not choose to create directories for your application, you should make this change. It has implications in other parts of the code.

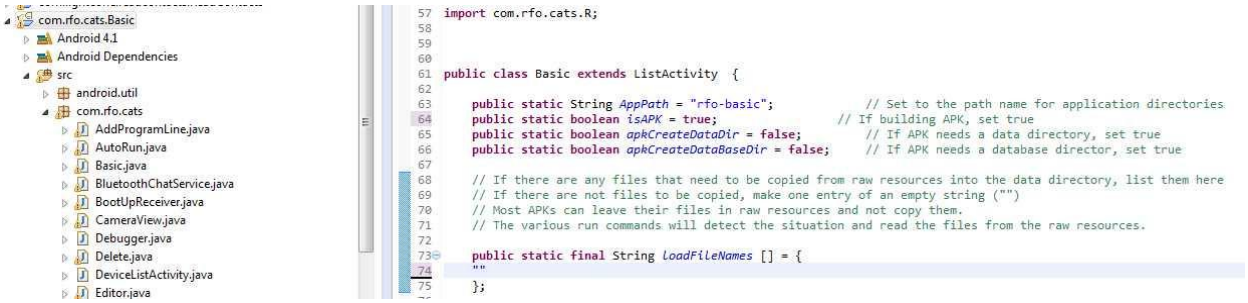
Change: *isAPK* = **false** to: *isAPK* = **true**;

Since this practice application does not need any directories, change:

apkCreateDataDir = **true**; -> *apkCreateDataDir* = **false**;

apkCreateDataBaseDir = **true**; -> *apkCreateDataBaseDir* = **false**;

Finally, we need to change the files to be loaded onto the SDCARD. Our practice application uses the file "meow.wav." We need to load this file onto the SDCARD for Standard Basic. We do not need to load it onto the SDARD for an APK. The APK will automatically read the file from res.raw. We do not need to load any files for this APK. To indicate that no files are to be loaded, remove them as shown below.



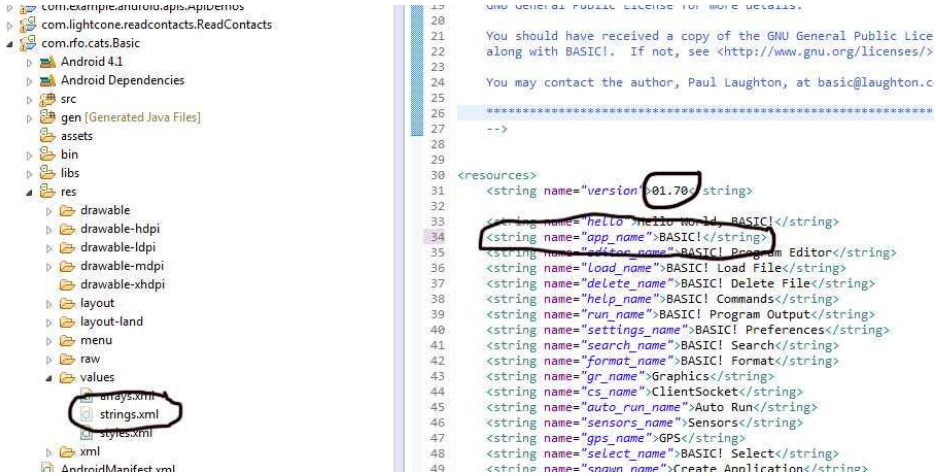
If your APK actually needs files to be loaded to the SDCARD, name them here. Be sure to allow the creation of the Data directory.

Files with the extension ".db" will be loaded into the /databases/ directory.

Save the changes to Basic.java.

Changing the APK Name

We are now going to change the name of the Application as it will appear on your Android device. Expand res.values and double click on strings.xml.



Change: <string name="*app_name*">BASIC!</string>

To: <string name="*app_name*">Cats</string>

Note: If you application uses the version$() function, this is where that function gets that value.

Save the changes.

Testing the APK

We are now ready to test this practice APK.

The first thing you will do is to create a Keystore. The Keystore is used to sign your application. Google Play requires this signing. Android devices will not install unsigned APKs. You will use this one Keystore for all your APKs. Preserve and protect it. You will not be able to update your APK without it.

For more information about the Keystore and signing, see:

http://developer.android.com/tools/publishing/app-signing.html

1. On the left hand side of the screen, right click on "com.rfo.cats.Basic"
2. Select: Android Tools->Export Signed Application Package
3. Select Next in the Project Checks dialog box.
4. Select "Create New Keystore."
    a. Provide a name and location for the Keystore.
    b. Provide a pass word and confirm it.
    c. Click next.

5. Fill out the Key Creation dialog.
    a. Pick any name for an Alias.
    b. Enter 25 for Validity (Years)
    c. Click next.
6. In the Destination and Key/Certificate Checks dialog,
    a. browse to the folder where you want to put the APK.
    b. Name the APK "Cats.apk"
    c. Click next.
7. Now, install and run Cats.apk

The APK will have the BASIC! icon. The name below the icon your device will be Cats. Double click Cats to run it.

If you have reached this point successfully then you are ready to customize the APK for your application.

Start over with a new copy of Basic.zip but use names and information particular to your application and then continue below.

Installing A BASIC! Program Into the Application

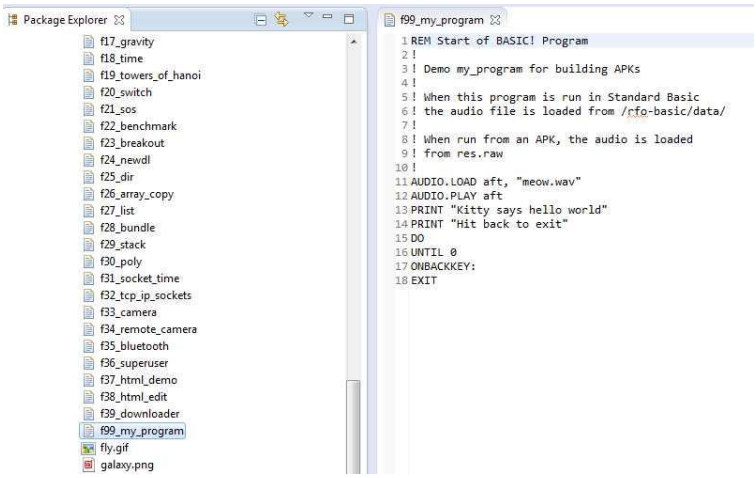
Outside of Eclipse

1. Open your BASIC! program in a text editor.
2. If your program uses INCLUDE files, you should text-merge the included files into a single text file.
3. Copy the entire program into the Clipboard.

In the Package Explorer on the left side

1. Open res.
2. Open raw.
3. Double click on f99_my_program.

The file will be opened



1. Tap inside the window on the right.
2. Tap Edit - > Select All
3. Tap Edit -> Paste
4. Tap the "X" on the f99_my_program tab to close the file.
5. Tap "Yes" on the Save Resource dialog box.

Adding Your Image and Audio Files

Looking into the Package Explorer under res.raw you will see a lot of files. The only one of these files that you will need in your APK is f99_my_app which we changed above. To delete the files, use shift-click to select the two blocks of files around f99_my_program and tap the delete key for each block. If you have done this correctly, you will have only f99_my_program left in res.raw.

You can now proceed to add your application specific image, audio or other files.

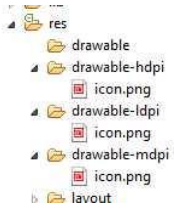
1. Outside of Eclipse, right click on the file you want import
2. Select copy
3. Back in Eclipse, right click on raw.
4. Select paste.

The file is now installed in your APK. When you open the file in your program, it will be read directly from res.raw.

Application ICONs

Android specifies that Application icons must be provided in three specific sizes: low dpi (36x36 pixels), medium dpi (48x48 pixels) and high dpi (72x72 pixels). The icons must also be .png files. There are tens of thousands of free icons available on the web. Most image editing programs can be used to re-sample the icons to the proper sizes and to save them as .png files. If you are not going to put your application on the Android market then you do not really need to worry about getting this exactly right.

To get your icon into your application, in res, open drawable-ldpi, drawable-mdpi, drawable-hdpi.



For each of the icon sizes:

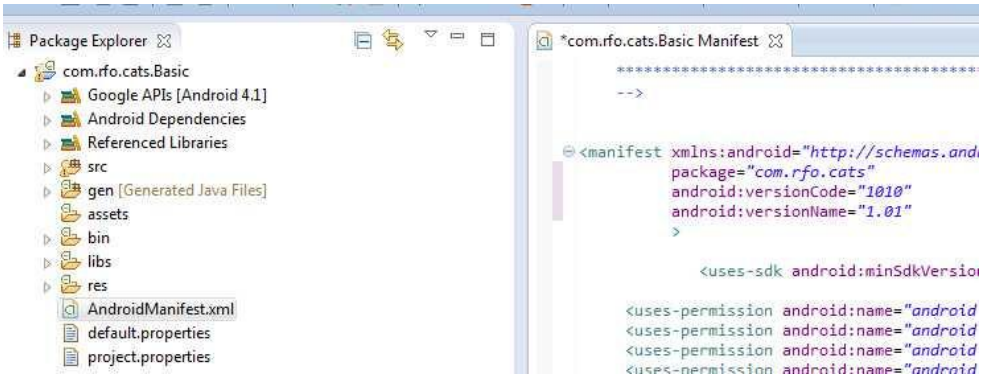
1. Outside of Eclipse, copy the icon file
2. In Eclipse, right click on the appropriate drawable- for the copied icon's size
3. Select Paste
4. Right click on the icon.png file and delete it
5. Select the newly pasted icon and rename it to "icon.png" by selecting File -> Rename.

Yes, it is tedious work.

Setting The Version Number and Version Name

If you are going to put the application on the Android Market, you will need to change the version number and name for each new release

Change the Versions information by double clicking on the AndroidManifest.xml file.



Make the appropriate changes to android:versionCode and android:versionName, click the X in the tab to close and save the changes.

If you want to use the BASIC! version$() function to have your program read your version number, you will also have to change the version number in res->values->strings.

Permissions

BASIC! uses many features about which the APK user is warned and must approve. Your particular APK may not need all or any of these permissions. The permission notifications are contained in the AndroidManiest.xml.

The permission notifications look like:

<uses-permission android.name="…….

Look them over. If you feel that your APK does not need them then delete or comment them out.

Please do keep the vibrate permission. If you do not have this permission, your APK may crash when it exits. This is what that permission looks like:

```
<uses-permission android:name="android.permission.VIBRATE"
android:required="false"></uses-permission>
```

If your application uses the SDCARD, do not comment out:

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

Be sure to test your APK after doing changing permission.

Preferences

There are certain preferences such as screen colors and font sizes that you have set for your application. The preferences that you will get with an APK will be BASIC! default preferences. You can change the default preferences if you wish.

Some preferences are simple check boxes. Other preferences are multiple-choice lists. The one check box preference whose default value that you might wish to change is the Console Lines preference. To change the default from lined console to unlined console:

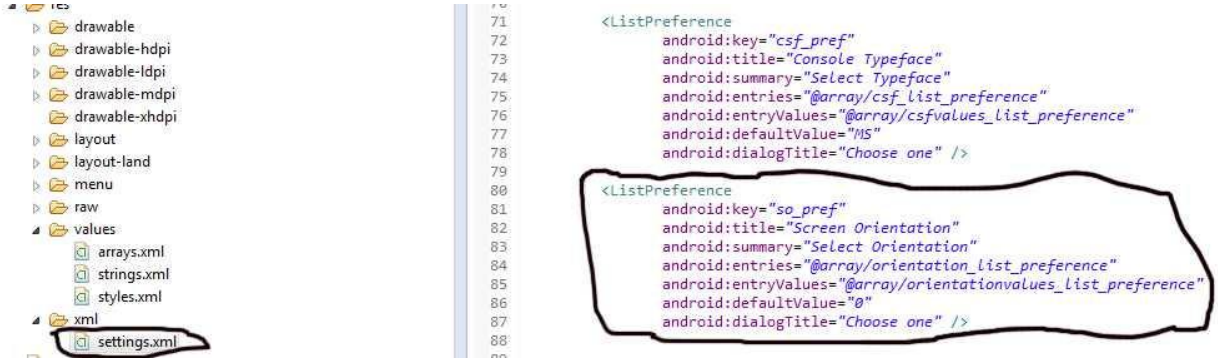
Open the res.xml hierarchy and double click on settings.xml. In the opened file scroll down to the indicated line and change the "true" to "false". Save the changes.

To change the multiple-choice preferences, open the res.values hierarchy and double click on arrays.xml. Each preference has two blocks. The top block lists the words that will be seen on the Android screen. The second block lists the internal names that correspond to the displayed words.

In the image below:

The section marked contains the names and values for the Screen Orientation preference. The top block is the display names. The bottom block is the internal values that correspond to the display name. For example the internal value of "Fixed Reverse Landscape" is 2.

To set a default value for Screen Orientation, we need to go back to settings.xml.



Find the block with the "android:title" (in the "Screen Orientation" section of the ListPreferences). That is the preference name that you see on the Android screen. The default value is in the "android:defaultValue =" line. Here we see the default value for the screen orientation is "0". Looking at the Array.xml file we can see the 0 is the internal name for "Variable By Sensors". To change the default value to "Fixed Reverse Landscape", change the 0 to 2.

The other list preferences follow the same logic.

Note: Be sure to test your application with your chosen preferences before burning them into the APK here.

Launch at device boot

Your APK can be set up to automatically launch just after the Android device has booted. This is accomplished by changing a parameter in AndroidManifest.xml.

Find the code line (around line 72):

```
<receiver android:enabled="false" android:name=".BootUpReceiver">
```

and change it to:

```
<receiver android:enabled="true" android:name=".BootUpReceiver">
```

Finished

Create your finished APK in the same way we created the practice APK.

Now that was not too bad, was it?

Appendix E – BASIC! Distribution License

BASIC! is distributed under the terms of the GNU General Public License which is reproduced here.

```
GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
```

can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If

the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains

in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this

License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to

copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this

License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible

for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have

actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

Apache Commons

Portions of BASIC! use Apache Commons.

Apache Commons Net

This product includes software developed by

The Apache Software Foundation (http://www.apache.org/).