De Re BASIC!

Version 1.57

June 19, 2012

Table of Contents

Changes in this Version 18

About the Title, De Re BASIC! 18

BASIC! Operation 18

- Editor 18
  - Editing the program 18
  - Menus 18
- Run 22
  - Menu 22
- Crashes 23

Command Description Syntax 23

- Upper and LowerCase 23
- <nexp> and <sexp> and <lexp> 23
- <var> and <var$> 23
- Array[] and Array$[] 23
- {something} 24
- X …,X 24
- {,n …,n} 24
- <statement> 24

Numbers 24

Strings 25

Variables 25

- Variable Names 25
- Variable Types 25
- Scalar and Array Variables 26
- Arrays 26
  - Array Commands 26

Data Structures and Pointers in BASIC! 29

- Lists 30
  - List Commands 30
- Bundles 32
  - Bundle Commands 33

Stacks 34
Stack Commands 34
Queues 35
Comments 36
! - Single Line Comment 36
!! - Block Comment 36
% - Middle of Line Comment 36
Expressions 36
Numeric <nexp> := {<numeric variable>|<numeric constant} {<noperator> <nexp>|<end of line>} 36
Numeric Operators <noperator> 36
Numeric Expression Examples 36
String <sexp> := {<string variable>|<string constant>} { + <sexp> | <end of line>} 36
Logical <lexp> 37
Logical Operators 37
Examples of Logical Expressions 37
Assignment Operations 37
Math Functions 38
BOR(<nexp1>, <nexp2>) 38
BAND(<nexp1>, <nexp2>) 38
BXOR(<nexp1>, <nexp2>) 38
ABS(<nexp>) 38
SQR(<nexp>) 38
RANDOMIZER(<nexp>) 38
RND() 39
CEIL(<nexp>) 39
FLOOR(<nexp>) 39
MOD(<nexp1>, <nexp2>) 39
ROUND(<nexp>) 39
LOG(<nexp>) 39
EXP(<nexp>) 39
SIN(<nexp>) 39
COS(<nexp>) 39

TAN(<nexp>) 39
TODEGREES(<nexp>) 39
TORADIANS(<nexp>) 39
ASIN(<nexp>) 39
ACOS(<nexp>) 40
ATAN(<nexp>) 40
VAL(<sexp>) 40
LEN(<sexp>) 40
HEX(<sexp>) 40
OCT(<sexp>) 40
BIN(<sexp>) 40
SHIFT (<value_nexp>, <bits_nexp>) 40
Clock() 40
ASCII(<sexp>) 40
Is_In(<Search_for_sexp>, <Search_in_sexp>{,<start_nexp>} 40
Starts_with (<Search_for_sexp>, <Search_in_sexp>{,<start_nexp>} 40
Ends_with (<Look_for_sexp>, <look_in_sexp>) 41
gr_collision (<object_1_nvar>, <object_2_navr>) 41
Background() 41
String Functions 41
CHR$ (<nexp>) 41
LEFT$ (<sexp>, <nexp>) 42
MID$(<sexp>, <start_nexp>, <Count_nexp>}) 42
REPLACE$(<target_sexp>, <argument_sexp>, <replace_sexp>) 42
RIGHT$(<sexp>, <nexp>) 42
STR$(<sexp>) 42
LOWER$(<sexp>) 42
UPPER$(<sexp>) 42
VERSION$() 42
HEX$(<nexp>) 43
OCT$(<nexp>) 43
BIN$(<nexp>) 43

FORMAT$(<Pattern_sexp>, <nexp>) 43
Leading Sign 43
Floating Field 43
Decimal Point 43
Pattern Character # 43
Pattern Character % 43
Overflow 43
Non pattern characters 43
Output Size 43
Examples: 44
User Defined Functions 44
Commands 44
Fn.def name|name$({nvar}|{svar}|Array[]|Array$[], ….. {nvar}|{svar}|Array[]|Array$[]) 44
Fn.rtn <sexp>|<nexp> 45
Fn.end 45
Call <user_defined_function> 45
Program Control Commands 45
If - Else -Elseif- Endif 45
For - To - Step - Next 46
F_n.break 47
While <lexp> - Repeat 47
W_r.break 47
Do - Until<lexp> 47
D_u.break 48
GoSub<label>, Return 48
GoTo <label> 48
Run <filename_sexp> {, <data_sexp>} 49
Switch Commands 49
Nesting Switch Operations 49
Sw.begin <nexp>|<sexp> 49
Sw.case <nexp>|<sexp> 50
Sw.break 50

Sw.default 50
Sw.end 50
OnError: 50
OnBackKey: 50
End 51
Exit 51
Debug Commands 51
Debug.on 51
Debug.off 51
Debug.echo.on 51
Debug.echo.off 51
Debug.print 51
Debug.dump.scalars 52
Debug.dump.array Array[] 52
Debug.dump.bundle <bundlePtr_nexp> 52
Debug.dump.list <listPtr_nexp> 52
Debug.dump.stack <stackPtr_nexp> 52
Console I/O 52
Output Console 52
CLS 52
Print <sexp>|<nexp> {,|;} . . . . <sexp>|<nexp>{,|;} 52
Console.save <filename_sexp> 53
User Input 53
Input <Prompt_sexp>, <nvar>|<svar>, {<Default_sexp>|<Default_nexp>} 53
Inkey$ <svar> 53
Text.input <savr>{, <sexp>} 54
TGet <result_svar>, <prompt_sexp> 54
Kb.show 54
Kb.hide 54
Working with Files 54
Paths Explained 54
File.Delete <Boolean_nvar>, <Path_sexp> 55

File.Dir <Path_sexp>, Array$[] 55
File.Exists <Boolean_nvar>, <Path_sexp> 55
File.Mkdir <Path_sexp> 56
File.Rename <Old_Path_sexp>, <New_Path_sexp 56
File.root <svar> 56
File.Size <size_nvar>, <Path_sexp> 56
Text File I/O 56
Text.open {r|w|a}, <File_table_nvar>, <Path_sexp> 56
Text.readln <File_table_nvar>, <Line_svar> 57
Text.writeln <File_table_nvar>, <sexp> 57
Text.position.get <File_table_nvar>, <position_nvar> 57
Text.position.set <File_table_nvar>, <position_nexp> 58
Text.close <File_table_nvar> 58
GrabURL <result_svar>,<url_sexp> 58
GrabFile <result_svar>,<path_sexp> 58
Byte File I/O 58
Byte.open {r|w|a}, <File_table_nvar>, <Path_sexp> 58
Byte.read.byte <File_table_nvar>, <byte_nvar> 59
Byte.write.byte <File_table_nvar>, <byte_nexp>|<sexp> 59
Byte.read.buffer <File_table_nvar>, <count_nexp>, <buffer_svar> 59
Byte.write.buffer <File_table_nvar>, <sexp> 59
Byte.position.get <File_table_nvar>, <position_nvar> 59
Byte.position.set <File_table_nvar>, <position_nexp> 60
Byte.copy <File_table_nvar>,<output_file_svar> 60
Byte.close <File_table_nvar> 60
HTML 60
Introduction 60
Commands 60
Html.open {<Show_status_bar_nexp>} 60
Html.load.url <file_sexp> 61
html.load.string <html_sexp> 61
Html.get.datalink <data_svar> 61

Html.go.back 62
Html.go.forward 62
Html.close 62
Html.clear.cache 62
HTML.clear.history 62
TCP/IP Sockets 62
TCP/IP Client Socket Commands 63
Socket.client.connect <server_ip_sexp>, <port_nexp> 63
Socket.client.read.ready <nvar> 63
Socket.client.read.line <line_svar> 64
Socket.client.read.file <fw_nexp> 64
Socket.client.write.line <line_sexp> 64
Socket.client.write.bytes <sexp> 64
Socket.client.write.file <fr_nexp> 64
TCP/IP Server Socket Commands 64
Socket.myip <svar> 64
Socket.server.create <port_nexp> 64
Socket.server.connect 65
Socket.server.read.ready <nvar> 65
Socket.server.read.line <svar> 65
Socket.server.write.line <sexp> 65
Socket.server.write.bytes <sexp> 65
Socket.server.write.file <fr_nexp> 65
Socket.server.disconnect 65
Socket.server.close 65
Socket.server.client.ip <nvar> 65
FTP Client 66
ftp.open <url_sexp>, <port_nexp>, <user_sexp>, <pw_sexp> 66
ftp.close 66
ftp.put <source_sexp>, <destination_sexp> 66
ftp.get <source_sexp>, <destination_sexp> 66
ftp.dir <list_navar> 66

ftp.cd <new_directory_sexp> 67
ftp.rename <old_filename_sexp>, <new_filename_sexp> 67
ftp.delete <filename_sexp> 67
ftp.rmdir <directory_sexp> 67
ftp.mkdir <directory_sexp> 67
Bluetooth 67
Bt.open 68
Bt.close 68
Bt.connect 68
Bt.reconnect 68
Bt.status <nvar> 68
Bt.write <sexp> 68
Bt.read.ready <navar> 68
Bt.read.bytes <svar> 68
Bt.device.name <svar> 69
Bt.set.uuid <sexp> 69
Miscellaneous Commands 69
Browse <url_sexp> 69
Clipboard 69
Clipboard.get <svar> 69
Clipboard.put <sexp> 69
Echo.on 69
Echo.off 69
Encryption 70
Encrypt <pw_sexp>, <source_sexp>, <encrypted_svar> 70
Decrypt <pw_sexp>, <encrypted_svar>, <decrypted_svar> 70
Text To Speech 70
Tts.init 70
Tts.speak <sexp> 70
Device <svar> 70
Include FileNamePath 70
Pause <ticks_nexp> 70

Popup <message_sexp>, <x_nexp>, <Y_nexp>, <duration_nexp> 70
Select <selection_nvar>, < Array$[]>|<list_nexp>, <message_sexp> {,<press_nvar>} 71
Split <result_Array$[]>, <source_sexp>, <test_sexp> 71
Time Year$, Month$, Day$, Hour$, Minute$, Second$ 72
Tone <frequency_nexp>, <duration_nexp> 72
Vibrate <Pattern_Array[]>,<nexp> 72
WakeLock <code_nexp> 72
http.post url$, list, result$ 73
myPhoneNumber <svar> 73
Phone.call <sexpr> 73
Sms.send <number_sexp>, <message_sexp> 74
Email.send <recipient_sxep>, <subject_sexp>, <body_sexp> 74
Headset <state_nvar>, <type_svar>, <mic_nvar> 74
SQLITE 74
Overview 74
SQLITE Commands 74
sql.open DB_Pointer, DB_Name$ 74
sql.close DB_Pointer 75
sql.new_table DB_Pointer, DB_Name$, Table_Name$, C1$, C2$..,CN$ 75
sql.drop_table DB_Pointer, Table_Name$ 75
sql.insert DB_Pointer, Table_Name$, C1$, V1$, C2$, V2$,...,CN$, VN$ 75
sql.query Cursor, DB_Pointer, Table_Name$, Columns$, Where$, Order$ 76
sql.next Done, Cursor, C1V$, C2V$, .., CNV$ 76
sql.delete DB_Pointer, Table_Name$, Where$ 77
sql.update DB_Pointer, Table_Name$, C1$, V1$, C2$, V2$,...,CN$, VN$: Where$ 77
sql.exec DB_Pointer, Command$ 77
sql.raw_query Cursor, DB_Pointer, Query$ 77
Graphics 77
Introduction 77
The Graphics Screen and Graphics Mode 77
Display Lists 78
Drawing into Bitmaps 78

Colors 78
Graphics Setup Commands 79
gr.open alpha, red, green, blue {, ShowStatusBar {, Orientation}} 79
gr.color alpha, red, green, blue, fill 79
gr.set.stroke <nexp> 79
gr.orientation <nexp> 79
Gr.StatusBar.Show <nexp> 79
gr.render 80
gr.screen width, height 80
gr.scale x_factor, y_factor 80
gr.cls 80
gr.close 81
gr.front flag 81
gr.brightness <nexp> 81
Graphical Object Creation Commands 81
gr.line Object_number, x1, y1, x2, y2 81
gr.rect Object_number, left, top, right, bottom 81
gr.oval Object_number, left, top, right, bottom 82
gr.arc Object_number, left, top, right, bottom, start_angle, sweep_angle, fill_mode 82
gr.circle Object_number, x, y, radius 82
gr.set.pixels Object_number, Pixels[] {,x,y} 82
Gr.poly Object_number, List_pointer {,x,y} 83
Hide and Show Commands 83
gr.hide Object_number 83
gr.show Object_number 83
Touch Query Commands 83
gr.touch touched, x, y 83
gr.bounded.touch touched, left, top, right, bottom 84
gr.touch2 touched, x, y 84
gr.bounded.touch2 touched, left, top, right, bottom 84
Text Commands 84
gr.text.align type 84

gr.text.size n 84
gr.text.width <nvar>, <sexp> 84
gr.text.bold Boolean 84
gr.text.underline Boolean 84
gr.text.strike Boolean 84
gr.text.skew n 85
gr.text.draw Object_number, x, y, text$ 85
Bitmap Commands 85
Gr.bitmap.create bitmap_ptr, width, height 85
gr.bitmap.load bitmap_ptr, File_name$ 85
gr.bitmap.size bitmap_ptr, Width, Height 85
gr.bitmap.scale dest_ptr, src_ptr, Width, Height {, Smoothing) 86
gr.bitmap.delete bitmap_ptr 86
gr.bitmap.crop <new_bitmap_object_navar>, <source_bitmap_object_nexp>, <x_nexp>, <y_nexp>, <width_nexp>, <height_nexp> 86
gr.bitmap.save Object_ptr, “filename”{, <quality_nexp>} 86
gr.bitmap.draw Object_ptr, bitmap_ptr, x , w 87
gr.get.bmpixel bitmap_ptr, x, y, alpha, red, green, blue 87
gr.bitmap.drawinto.start Bitmap_Pointer 87
gr.bitmap.drawinto.end 87
Rotate Commands 87
gr.rotate.start angle, x, y{,<obj_nvar>} 87
gr.rotate.end {<obj_nvar>} 88
Camera Commands 88
gr.camera.shoot bm_ptr 88
gr.camera.autoShoot bm_ptr {,flash_ mode} 89
Gr.camera.manualShoot bm_ptr {,flash_ mode} 89
Miscellaneous Commands 89
gr.screen.to_bitmap bm_ptr 89
gr.get.pixel x, y, alpha, red, green, blue 89
gr.get.position Object_number, x, y 89
gr.save “filename” {,<quality_nexp>} 89

gr.modify Object_number, parameter_name$, {value|value$}.....90
gr.paint.get <object_nvar> 91
gr_collision (<object_1_nvar>, <object_2_navr>) 91
gr.clip <object_nvar>, <left_nexp>,<top_nexp>, <right_nexp>, <bottom_nexp>{,<RO_nexp>}.....91
gr.NewDL Array[].....92
Audio Interface 93
Introduction 93
The Audio Interface.....93
Audio File Types 93
Commands 93
audio.load <aft_nvar>, <filename_sexp>.....93
audio.play <aft_nexp> 93
audio.stop 93
audio.pause.....93
audio.loop 94
audio.volume <left_nexp>, <right_nexp>.....94
audio.position.current <nvar> 94
audio.position.seek <nexp> 94
audio.length <length_nvar>, <aft_nexp>.....94
audio.release <aft_nexp> 94
audio.isdone <Boolean_nvar> 94
audio.record.start <fn_svar> 95
audio.record.stop.....95
SoundPool 95
Introduction 95
Commands 95
Soundpool.open <MaxStreams_nexp>.....95
Soundpool.load <soundID_nvar>, <file_path_sexp>.....95
Soundpool.unload <soundID_nexp>.....96
Soundpool.play streamID(nvar), soundID, right_volume, left_volume, priority, loop, rate 96
Soundpool.setvolume <streamID_nexp>, <leftVolume_nexp>, <rightVolume_nexp>.....96
Soundpool.setrate <streamID_nexp>, <rate_nexp>.....96

Soundpool.setpriority <streamID_nexp>, <priority_nexp> 96
Soundpool.pause <streamID_nexp> 96
Soundpool.resume <streamID_nexp> 96
Soundpool.stop <streamID_nexp> 96
Soundpool.release 97
GPS 97
Commands 97
gps.open 97
gps.close 97
gps.provider <svar> 97
gps.accuracy <nvar> 97
gps.latitude <nvar> 97
gps.longitude <nvar> 97
gps.altitude <nvar> 98
gps.bearing <nvar> 98
gps.speed <nvar> 98
gps.time <nvar> 98
Sensors 98
Introduction 98
Sensor Commands 99
sensors.list list$[] 99
sensors.open t1 {t2,...,tn} 99
sensors.read sensor_type, p1, p2, p3 99
sensors.close 99
Superuser 100
Commands 100
Su.open 100
Su.write <sexp> 100
Su.read.ready <nvar> 100
Su.read.line <svar> 100
Su.close 100
Appendix A - Command List 101

Appendix B – Sample Programs 109
f00a_download_manual.bas 109
f00b_basic_forum.bas 109
f02_arrays.bas 110
f03_goto_gosub.bas 112
f04_if_else.bas 113
f05_do_while.bas 115
f06_for_next.bas 117
f07_print_format.bas 118
f08_read_write_data.bas 119
f09_sql.bas 120
f10_graphics_object.bas 125
f11_graphics_touch.bas 128
f12_graphics_text.bas 129
f13_animations.bas 131
f14_compass.bas 134
f15_gps.bas 135
f16_select.bas 136
f17_gravity.bas 137
f18_time.bas 138
f19_towers_of_hanoi.bas 139
f20_switch.bas 144
F21_sos.bas 144
f22_benchmark.bas 145
F23_breakout.bas 151
f24_newdl.bas 158
f25_dir.bas 160
f26_array_copy.bas 162
f27_list.bas 163
F28_Bundles.bas 166
f29_stack.bas 167
f30_poly.bas 168

f31_socket_time 170
f32_tcp_ip_socket.bas 171
f33_camera.bas 174
f34_remote_camera.bas 176
f35_bluetooth.bas 182
f36_superuser.bas 185
f37_html_demo.bas 185
htmldemo1.html 187
htmlDemo2.html 191
F38_html_edit.bas 192
f39_downloader.bas 193
Appendix C – Launcher Short Cuts Tutorial 196
Introduction 196
The procedure for making a Shortcut Application 196
What you need to know 197
Appendix D – Building a Standalone Application 199
Introduction 199
License Information 199
Setting Up the Development Environment 199
Download the Stand Alone Application Source Code 199
Create a New Project in Eclipse 200
Cleaning the Project 201
Rename the package 202
Installing A BASIC! Program Into the Application 203
Making The Changes To Basic.java 204
Open the Basic.java file 204
Naming the application top level file directory 204
Creating Image and Audio Files 205
Application ICONs 207
Naming The Application 207
Setting The Version Number and Name 208
Warnings 209

Permissions 209
Compiling And Testing The Application 209
Creating An .apk File 210
Finished 211
Appendix E – BASIC! Distribution License 212
Apache Commons 223

Changes in this Version

The html.open command has been modified to accept an option parameter which requests the status bar to be shown. See page 60.

Running an empty program (a program with no executable statements) will now show the message, “Nothing to execute.” See page 22.

The new command, Run, causes a different BASIC! program file to be loaded and run. See page 49.

New line characters may now be inserted in string using “\n” See page 25

The gr.bitmap.scale command now has an optional parameter that provides for the optional smoothing of the scaled bitmap. See page 86.

The crashes and other strange behavior that had been occurring when a program was quickly rerun after stopping the program run with the Break key has been fixed.

About the Title, De Re BASIC!

“De Re” is Latin for “of the thing” or “about”.

BASIC! Operation

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.

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 this is the case for you then you will see different things depending upon the way you are holding the device. If you are holding the device in the 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 chuck 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 this mode. Hold your device in portrait mode.

If you do a long touch on the screen, a dialog box will appear. You can use the selections in the box for doing the selecting, copying, cutting and pasting of text … among other things.

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 to 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 files 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 pressed. 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 the "/source/" 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 then the tap will be ignored.

If you accidentally press the Load button, you can back out of Load by pressing 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 press 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 pressing 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 pressed at this point then the Editor will returned to with the found text selected.

If the **Replace Button** is pressed 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 pressed 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 and selected 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 back 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 pressed 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->Delete

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

Pressing 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 will be shown. Press the Delete button to delete the file. Press 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. Press the Delete button to delete the directory. Press the No button to dismiss to dismiss the dialog box and not delete the directory.

Pressing the ".." at the top of the screen move up one directory. 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 pressing 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.

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 orientation's apply to Android 2.3 or newer.

More -> Commands

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

Taping on a particular command causes that command to be copied to the clipboard (minus the page number) and exiting back 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

Pressing the Home key while in BASIC! leaves BASIC! in exactly the same state it was in when the Home key was pressed. 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.

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

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 the program prints something or the END statement is executed or you are in Echo mode or 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.

Pressing the BACK key will stop a running program. Pressing 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. Pressing 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 that created this crash. You can also give your email address if you would like the developer to contact you about the problem. You may also press Cancel and not report the problem.

The report sent 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 LowerCase

Commands are described using both upper a lower case for ease of reading. It does not matter if you use upper and lower case. BASIC! converts every character (except those in Quotes) to lower case when the program is run.

<nexp> and <sexp> and <lexp>

These notions denote a numeric expression (<nexp>) or a string expression<sexp> or a logical expression <lexp> expression. The expression can be a variable, a number, a quoted string or full expressions (a*x^2 + bx _ c).

<var> and <var$>

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

Array[] and Array$[]

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

{something}
Indicates something optional.

{ A | B |C }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 things separated by commas. At least one item is required.

{,n …,n}
Indicates an optional list of things with zero or more things 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:

- Printed number will always have decimal point. For example, 99 print as “99.0”.You can print numbers without decimal points by using the Format command. For example format(“##”,99) will print as “99”.
- Number with more the 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.

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>) command.

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 the characters “a” through “z”, or “_” or “#” or “@”. The remaining characters in the variable name may include the numbers 0 through 9.

A variable name may be as long as needed.

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

Variable Types

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

“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 inside the square brackets is called the index.

If you wanted to keep a list of ten animals, you could use and 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 as list for a “Cat” might be “Purrs” and “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. Iindex=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 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 in one way or another.

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 of 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. The extra, empty elements will be added to the start of the array if the optional minus(-) sign is present. If minus is not present then the extra, empty elements will be added to end of the array.

The array 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 list of values.

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.

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.

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 draw with folders in it. That file cabinet is maintained by your administrative assistant. You never see the file draw 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 draw. 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 (Pointer) for that information object. You then use that number (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 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 object. 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 data base in BASIC! you would 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. It 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 string type list will be created if the parameter is “S”. A numeric type list 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>}

Adds values to the <pointer_nexp> List. with the value, <nexp>{,<nexp>..,<nexp>}, of valuesThe array must not have been previously dimensioned.

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.

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.

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 (Numeric of String) 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 is a list of numbers.

List.size <pointer_nexp>, <svar>

The size of theList 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 now be zero.

List.search <pointer_nexp>, value|value$, <result_nvar>

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.

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 numeric pointer to the empty bundle. The fact that the bundle is represented by a numeric 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_nvar>, <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_nvar>, <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.type <pointer_nvar>, <key_sexp>, <type_svar>

Returns the value type (String or Numeric) of the specified key in the specified string variable. The <typr_svar> will contain an uppercase "N" if the type is numeric. The <typr_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 and 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 in <ptr_nvar>.

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

Pushes the <nexp> or <sexp> onto 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-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 (String or Number) 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.

Unary +, Unary –
Exponent ^
Multiply *, Divide /
Add +, Subtract –

Numeric Expression Examples

a
a*b + 4/d – 2*(d^2)
a + b + d + rnd(0)
b + ceil(d/25) + 5

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 expression. Both types produce a numerically represented 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.

Unary Not !
Less than "<", Greater than">", Less than or equal"<=", Greater than or equal ">="
Equal"=", Not Equal"<>"
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>

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.

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”

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. The floating point doubles 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. The floating point doubles 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. The floating point doubles 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>.

RANDOMIZER(<nexp>)

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

The Randomizer() function always returns zero.

If the numeric expression = 0 then the generator will be created using a random (time of day) seed.

If the numeric expression <> 0 then the generator will be created using the expression value as the seed. Random number generated with a non-zero seed will always be the same set of random 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 random number generator. If a Randomizer () has not been previously executed then a new random generator will be created using “Randomizer(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>.

EXP(<nexp>)

Returns e raised to the <nexp> power.

SIN(<nexp>)

Returns the trigonometric sine of angle <nexp>.

COS(<nexp>)

Returns the trigonometric cosine of angle <nexp>.

TAN(<nexp>)

Returns the trigonometric tangent of angle <nexp>.

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.

ACOS(<nexp>)

Returns the arc cosine of the angle, <nexp>, in the range of 0.0 through pi.

ATAN(<nexp>)

Returns the arc tangent of the angle <nexp>, in the range of -pi/2 through pi/2.

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 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 millisconds since the last boot.

ASCII(<sexp>)

Returns the ASCII value of the first character of <sexp>. If <sexp> is an empty string (“”) the value returned will be 256.

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

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

If the <Search_in_sexp> starting at <start_nexp> starts with (<Search_for_sexp> then the length of <Search_for_sexp> will be return 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> does not end with <Look_for_sexp> 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> starts. The value will always be greater or equal to 1.

gr_collision (<object_1_nvar>, <object_2_navr>)

The object <nvars> are the object's 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 hit. 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

CHR$ (<nexp>)

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

The character "C" is hexadecimal 43,

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 chars starting at Start.

<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 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$(<sexp>)

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. If the number is >= 0 then the first character will be a blank.

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 dictates the number of decimal digits that will be output.

Pattern Character

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

Pattern Character %

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

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 particularly ahead of 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 by the amount of memory that your particular Android device allocates to application.

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 from zero to dozens. 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 type 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 of either the call by value or the call by value type. 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 of the call by reference type. If there is no “&” in front of the calling variable name then the variable will be of the call by value type.

```
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 type.

```
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 in the main program named v$ is not the same 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 and when 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 - 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 loops can be nested to any level. When For loops are nested, any executed Next statement will apply to inner most executing For 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 immediatelyfollowing 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 may not be executed at all if <lexp> starts off true.

While loops may be nested to any level. When while loops are nested, any executed Repeat statement will apply to inner most While loop.

W_r.break

The While/Repeat loop will be terminated if this statement is executed and 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 loops may be nested to any level. When Do loops are nested, any executed Repeat statement will apply to inner most Do loop.

D_u.break

The Do/Until loop will be terminated if this statement is executed and 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>.

The statements at that location 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>.

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
```

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, pressing the back key will display the original program in the BASIC! editor.

Switch Commands

The Switch Commands are 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.

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 OneError: will make the program difficult to debug.

The OnError: label must stand alone on the line.

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 stop 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.

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.

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. Scalars 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

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 expressions.

If the semicolon (;) separator is used then nothing will separate the expressions printed.

If the semicolon is at the end of the line, the output will not be printed until Print statement 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>.

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 presses 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 an "OnError:" statement, execution will resume at the statement following the "OnError:" statement.

Inkey$ <svar>

Reports key presses 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 pressed, the string "key nn" will be returned. Where nn will be the Android key code for that key. If no key has been pressed, the "@" character is returned in Key$.

Text.input <savr>{, <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, press the Finish button. The edited text will be returned in <svar>.

If the BACK key is pressed 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 hits enter. The characters that user typed in will be returned in the result string variable. The prompt and response will be displayed on the Output Console.

Kb.show

Exposes the soft keyboard for use in inputting text to the running program. Keys may be read using the Inkey$ command. The command may not work in devices with hard or slide out keyboards.

Kb.hide

Hides the soft keyboard.

Working with Files

BASIC! can work with files anywhere on the SD Card. BASIC! does not access the 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 "/scard/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: "../../" The "../../" tells BASIC! to back up two levels from "/sdcard/rfo-basic/data" to "/sdcard/"

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

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

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

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

File.Delete <Boolean_nvar>, <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 <Boolean_nvar> will contain zero. If the file or directory did exist and was deleted, the <Boolean_nvar> 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 the Path string expression. 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 <Boolean_nvar>, <Path_sexp>

This command reports if the <Path_sexp> directory or file exists. If the directory or file does not exist, the <Boolean_nvar> will contain zero. If the file or directory does exist, the <Boolean_nvar> 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>.

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 or 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.

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 done reading a file, you should close it.

Text.writeln <File_table_nvar>, <sexp>

<sexp> will be written to the previously opened file represented by <File_table_nvar>. CR (Carriage Return) and NL (New Line) characters will be added to the end of <sexp>.

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.

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.

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 files (.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 text.read or 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.

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
Text.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 into the buffer string variable from the file. If the end of file has been reached, the length string, len(buf$), 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.

Byte.close <File_table_nvar>

Closes the previously opened file.

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 the 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 of these 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 better 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.

Html.load.url <file_sexp>

Loads and displays the file specified in the string. 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 hit the BACK key on the originally loaded page, the HTML viewer will be closed and the BASIC! output console will be displayed. If the originally loaded page links to another page then the BACK key is hit the it will be up 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.get.datalink <data_svar>

Gets the next datalink string from the datalink buffer. If there is no datalinked data available then the returned data will be and 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. 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 BACK key has been hit. The data is either "1" or "0"

If the data is "0" then the user hit BACK in the start screen. Going back is not possible therefor 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 hit the Submit button on a form with action=’FORM’ The form name/value pairs are returned. The html engine is automatically closed at this time. The BASIC! programmer may wish to open and new html session and display an html response page.

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 FTP/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 of 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 program, f32_tcp_ip_sockets.bas, demonstrate 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 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.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.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 can be repeatedly executed with timeouts.

Socket.client.read.file <fw_nexp>

A file transmitted by a server will be read and written to the file pointed to the file 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 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 to the previously connected Server 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.server.read.line is used to receive these bytes, the read.line command will not return until it receives a Line Feed (10, 0x0A) character.

Socket.client.write.file <fr_nexp>

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

```
Byte.open w, 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.

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.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 this command returns a non-zero value.

Socket.server.read.line <svar>

Reads a line sent from the previously connected Client and places the line into the line string variable. 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 can be repeatedly executed with timeouts

Socket.server.write.line <sexp>

Send the string expression 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 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 Line Feed (10, 0x0A) character.

Socket.server.write.file <fr_nexp>

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

```
Byte.open w, 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 sub directory of the current working directory, specify the path to that directory. For example, if there is a sub directory named "etc" then the filename, "/etc/name" would upload the file into that sub directory.

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 sub directory 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_navar>

Create a list of the files in the current working directory and place 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 an 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 Android "Settings" Application to enable Bluetooth and pair with any device(s) with which you plan to communicate.

When Bluetooth is opened (bt.open), it goes into the Listen Mode. While in this mode it will wait for a device to attempt to connect. You can change this mode to the Connect Mode by executing the bt.connect command.

Upon executing the bt.connect command you 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.state 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

Opens Bluetooth in Listen Mode. If you do not have Bluetooth enabled (using the the Android Settings Application) then you will be asked it you wish to enable Bluetooth.

Bt.close

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

Bt.connect

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.

Bt.reconnect

This command will attempt to reconnect to a device that was previously connected. The device must have been previously connect (during this Run) using the "bt.connect: command. The command cannot be used to reconnect to a device that was connected in the Listen mode.

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 <sexp>

Writes the string expression to the connected device. If the string expression is an empty string ("") then nothing will be written.

Bt.read.ready <navar>

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

Bt.read.bytes <svar>

The next available message is placed into the specified string variable. If there is not 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 and it won't clash. 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! UID 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, Office. The file types that ThinkFree Office can open are ".txt, .pdf, .doc,.xls, .rtf"

If your Android device does not already have ThinkFree Office on it, you can find ThinkFree Office in the Android Market. It is free.

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.

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:

Start the Settings Application
Select Voice input & output
Select Text-to-speech settings
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.

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_nvar>}

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 return.

< 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_nvar> is optional. If present, the type of the user press, Long or Short, will be returned in <press_nvar>. The value returned will be 0 if the user selected the item with a short tap. The value returned will 1 if the user selected the item with a long press.

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

The <spource_sexp> will be split into multiple strings which 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 for the specified duration in milliseconds.

The duration produced is does not exactly match the specified duration. If you need to get an exact duration, experiment.

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 then 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

Code	CPU	Screen	Keyboard Light
4	On	Bright	Bright

**If you hold a partial WakeLock, the CPU will continue to run, irrespective of any timers and even after the user presses 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 be 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$, list, result$

Execute a Post command to an Internet location.

url$ is a string expression giving the url ("http://....".) that will accept the Post.

List is a pointer to a string list which contains the Name/Value pairs needed for the Post.

Result$ is string variable into which 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 ambiguous.

Phone.call <sexpr>

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

Sms.send <number_sexp>, <message_sexp>

The SMS message in the string expression will be sent to number in the string expression. 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.

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: 1.0 if headset plugged in. 0 if no headset plugged in.

type: A string describing the device type.

mic: 1.0 if the headset has a microphone. 0 if the headset does not have microphone.

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, DB_Name$

Creates database for access. If the database does not exist, it will be created.

DB_Pointer: Is a pointer to the newly opened database. This value will be set by the sql.open command operation. DB_Pointer is used in subsequent database commands. DB_Pointer should not be altered.

DB_Name$: The filename used to hold this database in "data/data/com.rfo.basic/databases/"

Note: You may have more than one database/table opened at same time. Each opened database/table must have its own, distinct DB_Pointer variable.

sql.close DB_Pointer

Closes a previously opened database. The DB_Pointer value will be reset 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, DB_Name$, Table_Name$, 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$ in the referenced opened database. The column names for that table are defined by the following:

C1$, C2$,..,CN$: The names of the columns. 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 data base.

sql.drop_table DB_Pointer, Table_Name$

The table with Table_Name$ in the opened data base pointed to by DB_Pointer will be dropped from the data base if the table exists.

sql.insert DB_Pointer, Table_Name$, C1$, V1$, C2$, V2$,...,CN$, VN$

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

Table_Name$ 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, DB_Pointer, Table_Name$, Columns$, Where$, Order$

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

Columns$: A string with the list the names of the columns to be returned. The column names must be separated by commas. An example is Columns$ = " 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$: An SQL Expression string used to select which rows to return. In general, an SQL expression is of the form <Column Name> <operator> <Value>. For example, Where$ = "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.

The Order$ parameter specifies the order in which the rows are to be returned. The Order$ parameter selects 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, Cursor, 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. You must request the exact number of column values in this statement as you requested in the Query statement. 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 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 changes from 0 (false) to 1 (true). When Done 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, Table_Name$, Where$

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, Table_Name$, C1$, 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, Command$

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

sql.raw_query Cursor, DB_Pointer, Query$

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 than the BASIC! Text Output Screen. The Text Output Screen still exists and can still be written to. You can be returned to from 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 pressed. BASIC! will continue to run after the BACK key or MENU key is pressed when in Graphics Mode but the Output Console will be shown.

The BASIC! Output Console is hidden when the graphics screen is being displayed. Run time error message will not be observable. A Haptic feedback alert signals a run time error. This Haptic feedback will be a distinct, short buzzes. Hit 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 and infinite loop at the program end:

```
! 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 the 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 create animations without the overhead of having of recreating every object in the Object List.

Each time an object is added to the Object List, the Object Number is also added to the initial Display List. The user may optionally use the gr.NewDL 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 the gr.NewDL

Drawing into Bitmaps

You can draw into bitmaps in addition to drawing directly to 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.

Colors

Android colors consist of a mixture of Red, Green and Blue. Each of the colors can have 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 non-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 {, Orientation}}

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 is not zero.

The orientation upon opening graphics will be determined by the Orientation value. The Orientation values are the same as values for the gr.orientation command.

Note: The ShowStatusBar is optional; however, a ShowStatusBar value must be present in order to specify an Orientation value.

The default Orientation value (if Orientation is not specified) is zero (Landscape).

gr.color alpha, red, green, blue, fill

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

The BASIC! color command has an additional parameter, fill. The fill parameter is applied to enclosed objects such as circles or rectangles. If fill is false (0) then any enclosed object using that color will be displayed as an outline of the object. If fill is true (Not 0), then the enclosed object will be filled with the color.

gr.set.stroke <nexp>

Sets the line width of objects drawn after this command is issued. The <nexp> value must >=0. Zero produces the thinnest line and is the default stroke value.

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 display list has been fully displayed.

gr.screen width, height

Returns the screen's width and height in the variables. This command should be executed after (not before) any gr.orientation command. This is because the gr.orientation command swaps the height and width.

gr.scale x_factor, y_factor

Scale all drawing commands by the x and y scale factors. This command is provided to allow you to draw in a device independent manner and then scale the drawing 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} setting 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
Commands which screen, 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" while 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, 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 (x2,2). Returns the Object List object number for this line. This object will not be visible until the "gr.render" command is called.

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. 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. 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. 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. 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.

The 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 defines the point at which the polygon drawing start. Each subsequent coordinate defines a line drawn from the last coordinate to this coordinate.

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

The minimum number of coordinates 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.

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

Touch Query Commands

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.
```

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.

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.

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.text.bold Boolean

Makes the text bold if Boolean < >0 . If the color fill parameter is 0, only the outline of the bold test will be shown. If fill <>0, the text outline will be filled.

gr.text.underline Boolean

The drawn text will be underlined if Boolean <> 0.

gr.text.strike Boolean

The text will be drawn with a strike through line if Boolean <>0.

gr.text.skew n

Skews the text to give an italic effect. Negative values 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 n = -0.25.

gr.text.draw Object_number, x, y, text$

Creates and inserts a text object into the Object List. The text object will use the latest color and text preparatory commands. 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", "y", "text". The value for the "text" 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. ("/sdcard/Cougar.jpg")

"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.

Minus 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 numeric expression can be used request that the scaled image not be smoothed. If the expression is zero then the image will not be smoothed. If the optional parameter is not specified then the image will be smoothed.

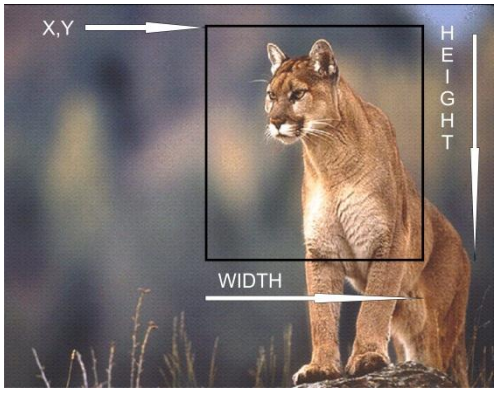
gr.bitmap.delete bitmap_ptr

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

gr.bitmap.crop <new_bitmap_object_navar>, <source_bitmap_object_nexp>, <x_nexp>, <y_nexp>, <width_nexp>, <height_nexp>

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

The <x_nexp>, <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 ".png"

The file will be saved as a PNG file if the filename if it does not end with either ".png" or ".jpg"

gr.bitmap.draw Object_ptr, bitmap_ptr, x , w

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. 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

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.

the width and height of the screen and must not be less than zero.

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 gr.rotate.start. 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 gr.rotate.end. 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 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 of these commands return pointers to bitmaps. 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 draw and render should be deleted using gr.bitmap.delete to avoid out-of-memory situations. You should also delete bitmaps if you are no longer going to draw and render them.

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 on using two different Android devices.

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.

The captured image is also stored as a file named "/sdcard/rfo-basic/data/image.png" If, for some reason, bm_ptr is return with zero, the image may still be able to be recovered from "image.png" One way of doing this would be:

```
gr.bitmap.load bm_ptr, "image.png"
```

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.

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.jpg"

Gr.camera.manualShoot bm_ptr {,flash_ mode}

This command is much like gr.camer.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.get.position Object_number, x, y

Get the current X,Y position of the specified display list object.

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 ".png"

The file will be saved as a PNG file if the filename if it does not end with either ".png" or ".jpg"

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 upon and PNG file. PNG files are always saved at the highest quality level.

Note: The file size of the JPEG file in inversely proportional to the quality. Lower quality values produce smaller files.

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.

As and examples, 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 and 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 lastest 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. Each command that draws something on the screen has a paint object associated with it. The paint objects contain information derived from gr.color and gr.text commands. Each time a gr.color or gr.text command is executed and new paint object is created. The new paint object contains the information from the last created paint object as modified by the current gr.color or gr.text command. The last created paint object is the paint object associated with a draw object when a draw command is executed.

The gr.paint.get command gets the object pointer of the last created paint object. This object pointer can be used to change the paint object associated the any previously draw object by means of the gr.modify command. The gr.modify parameter is "paint"

gr_collision (<object_1_nvar>, <object_2_navr>)

The object <nvar>s are the object's 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.

Note: gr_collision is a function, not a command.

gr.clip <object_nvar>, <left_nexp>,<top_nexp>, <right_nexp>, <bottom_nexp>{,<RO_nexp>}

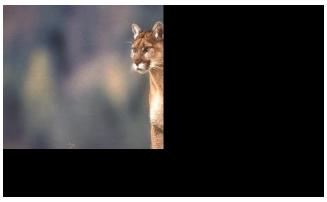
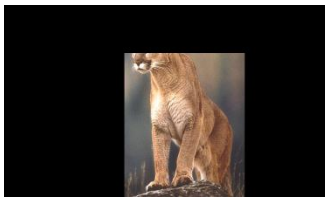
Clips objects drawn after this command is issued to be drawn only within the bounds of the specified clip rectangle.

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 the 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:

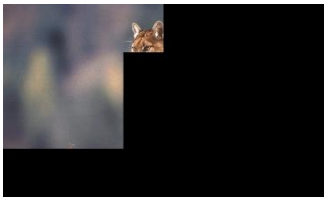
0	**Intersect**
1	**Difference**
2	**Replace**
3	**Reverse Difference**
4	**Union**
5	**XOR**

Examples:

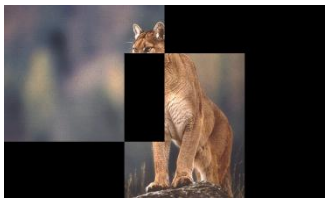
Original **Clip 1** **Clip 2**

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[]

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. You mileage may vary on these other file types.

Commands

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

audio.load <aft_nvar>, <filename_sexp>

Loads a music file into the Audio File Table. The Audio File Table index is returned in <aft_nvar>.

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

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 file playing when this command is executed. If the is file playing, execute audio.stop first.

The playing of the music stops 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

Stops playing of 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 then the file will re-start playing from the start 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 4 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 value 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 <Boolean_nvar>

If the current playing file is still playing then <Boolean_nvar> 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 simultaneously over top of each other. They can also be play over top of 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 are 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 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 being 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 specified file is loaded. Its sound ID is returned in the specified variable. 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/”

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

Soundpool.play streamID(nvar), soundID, right_volume, left_volume, priority, loop, rate
Starts the specified sound ID playing.

The stream ID is returned in the streamID numeric variable. 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 greater than or equal to zero. The lowest priority is zero.

The loop value of -1 will loop the sound 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 sound. 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>
Change the play rate of a 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 steams. 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 press 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 kph 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. Android's pre-defined Sensors are:

Accelerometer, Type = 1
Magnetic Field, Type = 2
Orientation, Type =3
Gyroscope, Type =4
Light, Type = Type = 5
Pressure, Type = 6
Temperature, Type = 7
Proximity, Type = 8
Gravity, Type = 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 your 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$[], must be an un-dimensioned array. The list will be returned in this array. The elements with contain the names and types of the available sensors. For example, "Gyroscope, Type =4". The following program snip can be used to list the elements of list$[].

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

The sensors.open 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 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 type parameter must be in the range of 0 to 9. 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 three parameters. 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 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, 36
!! - Block Comment, 36
% - Middle of Line Comment, 36
ABS(<nexp>), 38
ACOS(<nexp>), 40
Array.average <Average_nvar>, Array[], 27
Array.copy SourceArray[{<start>{,<length>}], DestinationArray[{{-}<extras>}, 27
Array.delete Array[], 27
Array.length <Length_nvar>, Array[], 27
Array.load Array[], <nexp>{,<nexp>..,<nexp>}, 27
Array.max <Max_nvar> Array[], 28
Array.min <Min_nvar>, Array[], 28
Array.reverse Array[] | Array$[], 28
Array.shuffle Array[] | Array$[], 28
Array.sort Array[] | Array$[, 28
Array.std_dev <sd_nvar>, Array[], 28
Array.sum <Sum_nvar>, Array[], 28
Array.variance <v_nvar>, Array[], 28
ASCII(<sexp>), 40
ASIN(<nexp>), 39
ATAN(<nexp>), 40
audio.isdone <Boolean_nvar>, 94
audio.length <length_nvar>, <aft_nexp>, 94
audio.load <aft_nvar>, <filename_sexp>, 93
audio.loop, 94
audio.play, 93
audio.play <aft_nexp>, 93
audio.position.current <nvar>, 94
audio.position.seek <nexp>, 94
audio.record.start <fn_svar>, 95
audio.record.stop, 95
audio.release <aft_nexp>, 94
audio.stop, 93
audio.volume <left_nexp>, <right_nexp>, 94
Background(), 41
BAND(<nexp1>, <nexp2>), 38
BIN$(<nexp>), 43
BIN(<sexp>), 40
BOR(<nexp1>, <nexp2>), 38
Browse <url_sexp>, 69
Bt.close, 68
Bt.connect, 68
Bt.device.name <svar>, 69
Bt.open, 68

Bt.read.bytes <svar>, 68
Bt.read.ready <navar>, 68
Bt.set.uuid <sexp>, 69
Bt.status <nvar>, 68
Bt.write <sexp>, 68
Bundle.clear <pointer_nvar>, 34
Bundle.create <pointer_nvar>, 33
Bundle.get <pointer_nvar>, <key_sexp>, <nvar>|<svar>, 33
Bundle.keys <pointer_nvar>, <list_nvar>, 33
Bundle.put <pointer_nexp>, <key_sexp>, <value_nexp>|<value_sexp>, 33
Bundle.type <pointer_nvar>, <key_sexp>, <type_svar>, 34
BXOR(<nexp1>, <nexp2>), 38
Byte.close <File_table_nvar>, 60
Byte.copy <File_table_nvar>,<output_file_svar>, 60
Byte.open {r|w|a}, <File_table_nvar>, <Path_sexp>, 58
Byte.position.get <File_table_nvar>, <position_nexp>, 59
Byte.position.set <File_table_nvar>, <position_nexp>, 60
Byte.read.buffer <File_table_nvar>, <count_nexp>, <buffer_svar>, 59
Byte.read.byte <File_table_nvar>, <byte_nvar>, 59
Byte.write.buffer <File_table_nvar>, <sexp>, 59
Byte.write.byte <File_table_nvar>, <byte_nexp>|<sexp>, 59
Call <user_defined_function>, 45
CEIL(<nexp>), 39
CHR$ (<nexp>), 41
Clipboard.get <svar>, 69
Clipboard.put <sexp>, 69
Clock(), 40
CLS, 52
Console.save <filename_sexp>, 53
COS(<nexp>), 39
D_u.break, 48
Debug.dump.array Array[], 52
Debug.dump.bundle <bundlePtr_nexp>, 52
Debug.dump.list <listPtr_nexp>, 52
Debug.dump.scalars, 52
Debug.dump.stack <stackPtr_nexp>, 52
Debug.echo.off, 51
Debug.echo.on, 51
Debug.off, 51
Debug.on, 51
Debug.print, 51
Decrypt <pw_sexp>, <encrypted_svar>, <decrypted_svar>, 70
Device <svar>, 70
Dim Array[<nexp>. . . ,<nexp>], 26
Do - Until<lexp>, 47
Echo.off, 69
Echo.on, 69
Email.send <recipient_sxep>, <subject_sexp>, <body_sexp>, 74

Encrypt <pw_sexp>, <source_sexp>, <encrypted_svar>, 70
End, 51
Ends_with (<Look_for_sexp>, <look_in_sexp>), 41
Exit, 51
EXP(<nexp>), 39
F_n.break, 47
File.Delete <Boolean_nvar>, <Path_sexp>, 55
File.Dir <Path_sexp>, Array[], 55
File.Exists <Boolean_nvar>, <Path_sexp>, 55
File.Mkdir <Path_sexp>, 56
File.Rename <Old_Path_sexp>, <New_Path_sexp, 56
File.root <svar>, 56
File.Size <size_nvar>, <Path_sexp>, 56
FLOOR(<nexp>), 39
Fn.def name|name$({nvar}|{svar}|Array[]|Array$[], ….. {nvar}|{svar}|Array[]|Array$[]), 44
Fn.end, 45
Fn.rtn <sexp>|<nexp>, 45
For - To - Step - Next, 46
FORMAT$(<Pattern_sexp>, <nexp>), 43
ftp.cd <new_directory_sexp>, 67
ftp.close, 66
ftp.delete <filename_sexp>, 67
ftp.dir <list_navar>, 66
ftp.get <source_sexp>, <destination_sexp>, 66
ftp.mkdir <directory_sexp>, 67
ftp.open <url_sexp>, <port_nexp>, <user_sexp>, <pw_sexp>, 66
ftp.put <source_sexp>, <destination_sexp>, 66
ftp.rename <old_filename_sexp>, <new_filename_sexp>, 67
ftp.rmdir <directory_sexp>, 67
GoSub<label>, Return, 48
GoTo <label>, 48
gps.accuracy <nvar>, 97
gps.altitude <nvar>, 98
gps.bearing <nvar>, 98
gps.close, 97
gps.latitude <nvar>, 97
gps.longitude <nvar>, 97
gps.open, 97
gps.provider <svar>, 97
gps.speed <nvar>, 98
gps.time <nvar>, 98
gr.arc Object_number, left, top, right, bottom, start_angle, sweep_angle, fill_mode, 82
Gr.bitmap.create bitmap_ptr, width, height, 85
gr.bitmap.crop <new_bitmap_object_navar>, <source_bitmap_object_nexp>, <x_nexp>, <y_nexp>,
   <width_nexp>, <height_nexp>, 86
gr.bitmap.delete bitmap_ptr, 86
gr.bitmap.draw Object_ptr, bitmap_ptr, x , w, 87
gr.bitmap.drawinto.end, 87

gr.bitmap.drawinto.start Bitmap_Pointer, 87
gr.bitmap.load bitmap_ptr, File_name$, 85
gr.bitmap.save Object_ptr, “filename”{, <quality_nexp>}, 86
gr.bitmap.scale dest_ptr, src_ptr, Width, Height {, Smoothing}, 86
gr.bitmap.size bitmap_ptr, Width, Height, 85
gr.bounded.touch touched, left, top, right, bottom, 84
gr.bounded.touch2 touched, left, top, right, bottom, 84
gr.brightness <nexp>, 81
gr.camera.autoShoot bm_ptr {,flash_ mode}, 89
Gr.camera.manualShoot bm_ptr {,flash_ mode}, 89
gr.camera.shoot bm_ptr, 88
gr.circle Object_number, x, y, radius, 82
gr.clip <object_nvar>, <left_nexp>,<top_nexp>, <right_nexp>, <bottom_nexp>{,<RO_nexp>}, 91
gr.close, 81
gr.cls, 80
gr.color alpha, red, green, blue, fill, 79
gr.front flag, 81
gr.get.bmpixel bitmap_ptr, x, y, alpha, red, green, blue, 87
gr.get.pixel x, y, alpha, red, green, blue, 89
gr.get.position Object_number, x, y, 89
gr.hide Object_number, 83
gr.line Object_number, x1, y1, x2, y2, 81
gr.modify Object_number, parameter_name$, {value|value$}, 90
gr.NewDL Array[], 92
gr.open alpha, red, green, blue {, ShowStatusBar {, Orientation}}, 79
gr.orientation 0|1, 79
gr.oval Object_number, left, top, right, bottom, 82
gr.paint.get <object_nvar>, 91
Gr.poly Object_number, List_pointer {,x,y}, 83
gr.rect Object_number, left, top, right, bottom, 81
gr.render, 80
gr.rotate.end {<obj_nvar>}, 88
gr.rotate.start angle, x, y{,<obj_nvar>}, 87
gr.save “filename” {,<quality_nexp>}, 86, 89
gr.scale x_factor, y_factor, 80
gr.screen width, height, 80
gr.screen.to_bitmap bm_ptr, 89
gr.set.pixels Object_number, Pixels[] {,x,y}, 82
gr.set.stroke <nexp>, 79
gr.show Object_number, 83
Gr.StatusBar.Show <nexp>, 79
gr.text.align type, 84
gr.text.bold Boolean, 84
gr.text.draw Object_number, x, y, text$, 85
gr.text.size n, 84
gr.text.skew n, 85
gr.text.strike Boolean, 84
gr.text.underline Boolean, 84

gr.text.width <nvar>, <sexp>, 84
gr.touch touched, x, y, 83
gr.touch2 touched, x, y, 84
gr_collision (<object_1_nvar>, <object_2_navr>), 41, 91
GrabFile <result_svar>,<path_sexp>, 58
GrabURL <result_svar>,<url_sexp>, 58
Graburl ip$, “http://automation.whatismyip.com/n09230945.asp”, 64
Headset <state_nvar>, <type_svar>, <mic_nvar>, 74
HEX$(<nexp>), 43
HEX(<sexp>), 40
Html.clear.cache, 62
HTML.clear.history, 62
Html.close, 62
Html.get.datalink <data_svar>, 61
Html.go.back, 62
Html.go.forward, 62
html.load.string <html_sexp>, 61
Html.load.url <file_sexp>, 61
Html.open {<Show_status_bar_nexp>}, 60
http.post url$, list, result$, 73
If - Else -Elseif- Endif, 45
Include FileNamePath, 70
Inkey$ <svar>, 53
Input <Prompt_sexp>, <nvar>|<svar>, {<Default_sexp>|<Default_nexp>}, 53
Is_In(<Search_for_sexp>, <Search_in_sexp>{,<start_nexp>}, 40
Kb.hide, 54
Kb.show, 54
LEFT$ (<sexp>, <nexp>), 42
LEN(<sexp>), 40
List.add <pointer_nexp>, <nexp>{,<nexp>..,<nexp>}, 30
List.add.array <destination_list_pointer_nexp>,Array$[]|Array[], 31
List.add.list <destination_list_pointer_nexp>, <source_list_pointer_nexp>, 31
List.clear <pointer_nexp>, 32
List.create <N|S>, <pointer_nvar>, 30
List.create N|S, <pointer_nvar>, 30
List.get <pointer_nexp>,<index_nexp>, <svar>|<nvar>, 32
List.insert <pointer_nexp>,<index_nexp>, <sexp>|<nexp>, 31
List.remove <pointer_nexp>,<index_nexp>, 31
List.replace <pointer_nexp>,<index_nexp>, <sexp>|<nexp>, 31
List.search <pointer_nexp>, value|value$, <result_nvar>, 32
List.size <pointer_nexp>, <svar>, 32
List.ToArray <pointer_nexp>, Array$[] | Array[], 32
List.type <pointer_nexp>, <svar>, 32
LOG(<nexp>), 39
LOWER$(<sexp>), 42
MID$(<sexp>, <start_nexp>, <Count_nexp>}), 42
MOD(<nexp1>, <nexp2>), 39
myPhoneNumber <svar>, 73

OCT$(<nexp>, 43
OCT(<sexp>), 40
OnBackKey:, 50
OnError:, 50
Pause <ticks_nexp>, 70
Phone.call <sexpr>, 73
Popup <message_sexp>, <x_nexp>, <Y_nexp>, <duration_nexp>, 70
Print <sexp>|<nexp> {,|;} . . . . <sexp>|<nexp>{,|;}, 52
RANDOMIZER(<nexp>), 38
REPLACE$(<target_sexp>, <argument_sexp>, <replace_sexp>), 42
RIGHT$(<sexp>, <nexp>), 42
RND(), 39
ROUND(<nexp>), 39
Run <filename_sexp> {, <data_sexp>}, 49
Select <selection_nvar>, < Array$[]>|<list_nexp>, <message_sexp> {,<press_nvar>}, 71
sensors.close, 99
sensors.list list$[], 99
sensors.open t1 {t2,...,tn}, 99
sensors.read sensor_type, p1, p2, p3, 99
SHIFT (<value_nexp>, <bits_nexp>), 40
SIN(<nexp>), 39
Socket.client.connect <server_ip_sexp>, <port_nexp>, 63
Socket.client.read.file <fw_nexp>, 64
Socket.client.read.line <line_svar>, 64
Socket.client.read.ready <nvar>, 63
Socket.client.write.bytes <sexp>, 64
Socket.client.write.file <fr_nexp>, 64
Socket.client.write.line <line_sexp>, 64
Socket.myip <svar>, 64
Socket.server.client.ip <nvar>, 65
Socket.server.close, 65
Socket.server.connect, 65
Socket.server.create <port_nexp>, 64
Socket.server.disconnect, 65
Socket.server.read.line <svar>, 65
Socket.server.read.ready <nvar>, 65
Socket.server.write.bytes <sexp>, 65
Socket.server.write.file <fr_nexp>, 65
Socket.server.write.line <sexp>, 65
Soundpool.load <soundID_nvar>, <file_path_sexp>, 95
Soundpool.open <MaxStreams_nexp>, 95
Soundpool.pause <streamID_nexp>, 96
Soundpool.play streamID(nvar), soundID, right_volume, left_volume, priority, loop, rate, 96
Soundpool.release, 97
Soundpool.resume <streamID_nexp>, 96
Soundpool.setpriority <streamID_nexp>, <priority_nexp>, 96
Soundpool.setrate <streamID_nexp>, <rate_nexp>, 96
Soundpool.setvolume <streamID_nexp>, <leftVolume_nexp>, <rightVolume_nexp>, 96

Soundpool.stop <streamID_nexp>, 96
Soundpool.unload <soundID_nexp>, 96
Split <result_Array$[]>, <source_sexp>, <test_sexp>, 71
sql.close DB_Pointer, 75
sql.delete DB_Pointer, Table_Name$, Where$, 77
sql.drop_table DB_Pointer, Table_Name$, 75
sql.exec DB_Pointer, Command$, 77
sql.insert DB_Pointer, Table_Name$, C1$, V1$, C2$, V2$,...,CN$, VN$, 75
sql.new_table DB_Pointer, DB_Name$, Table_Name$, C1$, C2$..,CN$, 75
sql.next Done, Cursor, C1V$, C2V$, .., CNV$, 76
sql.open DB_Pointer, DB_Name$, 74
sql.query Cursor, DB_Pointer, Table_Name$, Columns$, Where$, Order$, 76
sql.raw_query Cursor, DB_Pointer, Query$, 77
sql.update DB_Pointer, Table_Name$, C1$, V1$, C2$, V2$,...,CN$, VN$: Where$, 77
SQR(<nexp>), 38
Stack.clear <ptr_nexep>, 35
Stack.create N|S, <ptr_nvar>, 34
Stack.IsEmpty <ptr_nexep>, <nvar>, 35
Stack.peek <ptr_nexep>, <nvar>|<svar>, 35
Stack.pop <ptr_nexep>, <nvar>|<svar>, 35
Stack.push <ptr_nexep>, <nexp>|<sexp>, 34
Stack.type <ptr_nexep>, <svar>, 35
Starts_with (<Search_for_sexp>, <Search_in_sexp>{,<start_nexp>}, 40
STR$(<sexp>), 42
Su.close, 100
Su.open, 100
Su.read.line <svar>, 100
Su.read.ready <nvar>, 100
Su.write <sexp>, 100
Sw.begin <nexp>|<sexp>, 49
Sw.break, 50
Sw.case <numeric_constant>|<string_constant>, 50
Sw.default, 50
Sw.end, 50
TAN(<nexp>), 39
Text.close <File_table_nvar>, 58
Text.input <savr>{, <sexp>}, 54
Text.open {r|w|a}, <File_table_nvar>, <Path_sexp>, 56
Text.position.get <File_table_nvar>, <position_nvar>, 57
Text.position.set <File_table_nvar>, <position_nexp>, 58
Text.readln <File_table_nvar>, <Line_svar>, 57
Text.writeln <File_table_nvar>, <sexp>, 57
TGet <result_svar>, <prompt_sexp>, 54
Time Year$, Month$, Day$, Hour$, Minute$, Second$, 72
TODEGREES(<nexp>), 39
Tone <frequency_nexp>, <duration_nexp>, 72
TORADIANS(<nexp>), 39
Tts.init, 70

Tts.speak <sexp>, 70
UnDim Array[], 27
UPPER$(<sexp>), 42
VAL(<sexp>), 40
VERSION$(), 42
Vibrate <Pattern_Array[]>,<nexp>, 72
W_r.break, 47
WakeLock <code_nexp>, 72
While <lexp> - Repeat, 47

Appendix B – Sample Programs

These programs are loaded into "/sdcard/rfo-basic/source/ Sample Program" when a new release of BASIC! is installed.

You can force BASIC! to re-load these programs by:

Using 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!

f00a_download_manual.bas

```
!!
This command down loads the
BASIC! manual, De Re BASIC!

The manual is in PDF format
and can be read by any of the
PDF readers available on the
Android Market.

Most Android products come
with a PDF reader application. Thus
this program will attempt to open
the manual for reading.

The manual will be down loaded
to the SDCARD in the directory
"/sdcard/rfo-basic/data/De_Re_BASIC!.pdf"

This download will take up
to a minute. Be patient.

!!
byte.open r, fn, "http://laughton.com/basic/help/De_Re_BASIC!.pdf"
popup "Be patient. Wait for 'Download Done' Message", 0, 0, 1
byte.copy fn, "De_Re_BASIC!.pdf"
print "Download Done."
print "Opening the manual in your PDF reader."
browse "file:///sdcard/rfo-basic/data/De_Re_BASIC!.pdf"
end
```

f00b_basic_forum.bas

```
!!
Running this program will take
you to the BASIC! forum.

The BASIC! forum is where
BASIC! users gather to
share programs, help one
another, report bugs, and
suggest enhancements.
```

```
Join the forum and join
the BASIC! development
team.
!!
browse "http://rfobasic.freeforums.org/"
end
```

f02_arrays.bas

```
!!

This file contains executable examples
Array operations. Examine the code
and run then run this program.

Arrays must dimensioned (DIMed) before
the can be used. The DIM tells the
program how many lists or lists
of lists you will want.
!!

DIM array_1[10]
DIM array_2[10, 5], array_3[10,5,3]

! The following examples fills a
! numeric array with successive numbers
! and then prints the array.

print "Example 1"

dim able[2,3,4]
p = 0
for i=1 to 2
        for j=1 to 3
                for k=1 to 4
                        able[i,j,k] = p
                        p=p+1
                next k
        next j
next i

print "Array able[,,] loaded"

buffer$=""
for i=1 to 2
        for j=1 to 3
                for k=1 to 4
                        s$ = format$("##", able[i,j,k])
                        buffer$ = buffer$+ ", " + s$
                next k
                print i,j, buffer$
                buffer$=""
        next j
next i

! The following example fills a
! string array with successive values
```

```
! and then prints the array

print "Example 2"

dim able$[2,3,4]
p = 1
for i=1 to 2
        for j=1 to 3
                for k=1 to 4
                        able$[i,j,k] = format$("##",p)
                        p=p+1
                next k
        next j
next i

print "Array able$[] loaded"

buffer$=""
for i=1 to 2
        for j=1 to 3
                for k=1 to 4
                        buffer$ = buffer$+ ", " +able$[i,j,k]
                next k
                print i,j, buffer$
                buffer$=""
        next j
next i


! Array.xxxx Commands

!  Start of demos using numeric
! arrays

array.load x[], 3,5,6,9,1~
4,2,7,8
title$="Original load"
gosub ShowNumbers

array.sort x[]
title$="Sorted"
gosub ShowNumbers

array.reverse x[]
title$="Reversed"
gosub ShowNumbers

array.shuffle x[]
title$="Shuffled"
gosub ShowNumbers

! Skip over subroutine

goto skip

! Subroutine to show the
! numeric array values
```

```
ShowNumbers:
print title$
array.length length,x[]
for i=1 to length
print x[i];", ";
next i
print ""
return

skip:

!  Start of  numeric array information
!  extraction commands

array.sum sum, x[]
print "Sum = "; sum

array.average avg, x[]
print "Average = "; avg

array.min min, x[]
print "Min = "; min

array.max max, x[]
print "Max = "; max

! Start of string array demos

array.load z$[],"abc","def","ghi","jkl"~
"mno", "pqr", "stu", "vwx", "yz*"

print "Loaded string array"

array.length length,z$[]
for i=1 to length
print z$[i];" ";
next i
print ""

print "Reversed string array"
array.reverse z$[]

array.length length,z$[]
for i=1 to length
print z$[i];" ";
next i
print ""

end
```

f03_goto_gosub.bas

```
!!
This file contains executable examples
GoTo and GoSub operations. Examine
the code  and run then run this program.
!!
```

```
! GoTo Example

print "GoTo Example"

print 1
goto skip
print 1000
skip:
print 2

! Prints 1, 2

! GOSUB Example

print "GoSub Example"

print 1
gosub subroutine
print 4
print 5
goto done
subroutine:
print 2
print 3
RETURN
done:
print "done"

! Prints 1, 2, 3, 4, 5, done
```

f04_if_else.bas

```
!!
This file contains executable examples
if - then - else operations. Examine
the code  and run then run this program.
!!

! The code will print numbers
! 1 through 13 in the sequence
!
PRINT "Example 1"

IF 1<2 THEN PRINT 1 ELSE PRINT 1000

IF 1>2 THEN PRINT 2000 ELSE PRINT 1

IF 1 THEN
 PRINT 3
 PRINT 4
ELSE
 PRINT 3000
 ENDIF
PRINT 5

!
! Note: IF no statement is going to follow
```

```
! the THEN on same line,
! the THEN is assumed and
! need not be present
!

IF 0
 PRINT 4000
 PRINT 4001
ELSE
 PRINT 6
 ENDIF
PRINT 7

IF 1 | 0 THEN
 PRINT 8
 PRINT 9
ELSE
 PRINT 5000
 PRINT 5001
 ENDIF
PRINT 10

IF 1 & 0 THEN
 PRINT 6000
 PRINT 6001
ELSE
 PRINT 11
 PRINT 12
 ENDIF
PRINT 13

PRINT "Example 2"

IF 1 = 1 THEN
 PRINT 1
 IF 1 = 2 THEN
  PRINT 1000
 ELSE
  PRINT 2
  ENDIF
 PRINT 3
ENDIF

IF 3 <> 3 THEN
 PRINT 2000
 IF 1 = 2 THEN
  PRINT 1000
  ELSE
   PRINT 2000
   ENDIF
 PRINT 3000
ELSE
 PRINT 4
 PRINT 5
 ENDIF


IF 1 & 1
```

```
 PRINT 6
 IF 1 THEN
  PRINT 7
  IF 2 THEN
   PRINT 8
   PRINT 9
  ELSE
   PRINT 9000
   ENDIF
  PRINT 10
  ENDIF
 PRINT 11
ENDIF

! ENDIF provides for a chain
! of embedded IF statements.

! Note that only if first
! matching ELSEIF is executed,

! Note that the THEN is optional

PRINT "Example 3"

x=1
IF x=3
 PRINT 3
ELSEIF x=2 THEN
 PRINT 2
ELSEIF x=1
 PRINT 1
ELSEIF x=4 THEN
 PRINT 4
ELSEIF x=3
 PRINT 3000
ENDIF

! ESLEIF may be followed
! by and ELSE; however
! no following ELSEIF
! will ever be executed.

x=1
IF x=3
 PRINT 3
 ELSE IF x =4
  PRINT 4
  ELSE
   PRINT 2
 ELSIF x=1
  PRINT 1
ENDIF
```

f05_do_while.bas

```
!!
```

```
This file contains executable examples
DO - UNTIL and WHILE - REPEAT operations.
Examine the code  and run then run this program.
!!

! DO - UNTIL Example

! The following example will print
! 1, 2, 3, 3

PRINT "Do Example"
index = 1
limit = 3
DO
        PRINT index
        index = index + 1
UNTIL index = limit
PRINT index

! The following example will print
! 4, 5, 6, 6

PRINT "While Example"
index = 4
limit = 6
WHILE index <= limit
        PRINT index
        index = index + 1
        REPEAT
PRINT index
!
! Both loop types may be nested
! to any level
!
! The following example will print
!
PRINT "Nested DO and WHILE examples"

index_1 = 1
limit_1 = 3
DO
        index_2 = 1
        limit_2 = 3
                WHILE index_2 < limit_2
                        index_3 = 1
                        limit_3 = 3
                        WHILE index_3 < limit_3
                                PRINT index_1, index_2, index_3
                                index_3 = index_3 + 1
                                REPEAT
                        index_2 = index_2 + 1
                        REPEAT
                index_1 = index_1 + 1
        UNTIL index_1 = limit_1
PRINT index_1*10, index_2*10, index_3*10

END
```

f06_for_next.bas

```
!!
This file contains executable examples
For - Next operations. Examine
the code  and run then run this program.
!!

PRINT "Example 1"
FOR i = 1 TO 3 STEP 1
        PRINT i
        NEXT i
PRINT i * 10

! will print 1, 2, 3, 40

PRINT "Example 2"
For j = 1 TO 3
        PRINT j
        NEXT j
PRINT j*10

! will also print 1, 2 , 3, 40
!
! if the STEP is not present, STEP 1
! is assumed

PRINT "Example 3"
start = 1
stop = 3
FOR index = start TO stop STEP 2
        PRINT index
        NEXT index
PRINT index*10

! will print 1, 2, 50
!
! The loop will repeat until the
! index > stop
!
! If the STEP value is negative
! the loop will repeat until the
! index < stop

PRINT "Example 4"
start = 3
stop = 1
FOR index = start TO stop STEP -2
        PRINT index
        NEXT index
PRINT index*10

! will print 3, 1, -10
!
! FOR loops can be nested to any number of levels
!
PRINT "Example 4"
FOR i = 1 to 2
```

```
                FOR j = 1 TO 2
                        FOR k = 1 to 2
                                PRINT i, j, k
                        NEXT k
                NEXT j
        NEXT k

! The index variable name
! after NEXT is not required nor
! is it checked. It is essentially
! a comment to help you keep track
! your nesting.
!
! Some restrictions on variable names..
!
! FOR <var> = <exp1> TO <exp2> STEP <exp3>
!
! The variables used in <exp1> must not contain
! the character sequence TO.
!
! The variables used in <exp2> must not contain
! the character sequence STEP.
!
! If you do this, you will get unexpected results.

bottom = 2
top = 3
FOR index = bottom TO top  STEP 2
PRINT index
NEXT
```

f07_print_format.bas

```
!!
This file contains executable examples
of PRINT and FORMAT operations.
Examine the code  and run then run
this program.
!!

! PRINT

print "Comma seperated example"
a$ = "ABC"
b = 123
c$ = "DEF"
print a$, b, c$, "GHI"

print "Semicolon seperated example"
a1$ = "The quick "
a2$ = "brown fox "
print a1$;a2$

!print "Semicolon ata end of line example"
a3$ = "jumped over "
print a1$;a2$;
print a3$
```

```
! FORMAT

print "Format Example 1"
print format$("%%%%%.###", 123.456)
print format$("%%%%%.###", -123.456)
print format$("$%%%%%.###", 123.456)
print format$("$%%%%%.##", -123.456)
print format$("$%%%%%", -123.456)
print format$("#####.###", 123.456)
print format$("#####.###", -123.456)
print format$("$#####.###", 123.456)
print format$("$#####.##", -123.456)
print format$("$#####", -123.456)

print "Format Example 2"
print format$("$##,###,###,###", 123)
print format$("$##,###,###,###", 1234)
print format$("$##,###,###,###", 1234)
print format$("$##,###,###,###", 1234567)
print format$("$##,###,###,###", 12345678)


print "Format Example 3"
print format$("@ #/###/###/###", 1234567890)
print format$("@ ###/###", -1234567890)

END
```

f08_read_write_data.bas

```
!!
This file contains executable examples
of File I/O operations.
Examine the code  and run then run
this program.
!!
Print "********* Text  Read, Write Example"

text.open W, FN1, "testfile.txt"
text.writeln FN1, "Test line 1"
text.writeln FN1, "Test line 2"
text.close FN1

text.open R, FN2, "testfile.txt"
do
        text.readln FN2, a_line$
        print a_line$
until a_line$ = "EOF"
text.close FN2


! Byte file I/O

Print "********* Byte.copy example"
BYTE.OPEN R, rf, "Cartman.png"
```

```
BYTE.COPY rf, "Cartman2.png"
BYTE.CLOSE rf

print "******** Show Cartman2 exists"
file.exists b, "Cartman2.png"
if b then print "It does exist" else print "It does not exist"

Print "******** Sizes of the files"
file.size s1, "Cartman.png"
file.size s2, "Cartman2.png"
print s1, s2


! MKDIR and DIR Example

Print "******** MKDIR and DIR Example"

Print "mkdir \"newdir\""
file.mkdir "newdir"
file.dir "", dir$[]
array.length length, dir$[]

for index = 1 to length
        print dir$[index]
next index

! RENAME

Print "******** Rename Example"
Print "rename \"newdir\", \"olddir\""
file.rename "newdir", "olddir"
file.dir "", dir2$[]
array.length length, dir2$[]

for index = 1 to length
        print dir2$[index]
next index

Print "******** Root example"
file.root r$
print r$

END
```

f09_sql.bas

```
! An SQL Interface Example
!
! Ms O'Brien teaches Java at San Jose State.
! Her students are always asking what their
! overall grade average is. In order to
! quickly answer their question, she has
! created an SQL database with this
! information. She keeps the name (first,
! last) of the student, the current grade
```

```
! average, and the student's attendance
! percentage. In this example, she is
! creating the table for her Java 101
! class.
!
! First she created the database.

dbname$ = "grades.db"


sql.open DB_Ptr, dbname$

! Now that the database has been
! opened or created, she can use the returned
! DB_ptr to work on the database.
!
! The next thing she needs to do is create a
! table in this new database.

tbname$ = "java101"
c1$ = "first_name"
c2$ = "last_name"
c3$ = "grade_average"
c4$ = "percent_attendance"

! Before the table is created, it will be removed
! (Drop) the table from the database if it already
! exists. This will avoid duplicate rows in the
! table resulting from multiple executions of this
! program.

sql.drop_table DB_Ptr, tbname$

sql.new_table DB_Ptr, tbname$, c1$, c2$, c3$, c4$

! The next thing she needs to do is initialize
! the database table with some students and their
! data.

fn$ = "Betty"
ln$ = "Ng"
ga$ = "96"
pa$ = "100"
gosub insert

fn$ = "Janice"
ln$ = "Chen"
ga$ = "88"
pa$ = "100"
gosub insert

fn$ = "Bill"
ln$ = "Wilkinson"
ga$ = "43"
pa$ = "28"
gosub insert

fn$ = "Jose"
```

```
ln$ = "Martinez"
ga$ = "72"
pa$ = "65"
gosub insert

fn$ = "Tamasin"
ln$ = "Washington"
ga$ = "91"
pa$ = "96"
gosub insert

fn$ = "Shania"
ln$ = "Lewis"
ga$ = "82"
pa$ = "100"
gosub insert

fn$ = "Pierre"
ln$ = "Thomas"
ga$ = "78"
pa$ = "65"
gosub insert

goto skip1

! INSERT subroutine

insert:
        sql.insert DB_Ptr, tbname$, c1$,fn$,  c2$,ln$, c3$,ga$, c4$,pa$
return

! Now that she has built the database, she should
! look at the data base to insure everything
! is correct.
!
skip1:

title$ = "First look"
gosub show_all:

goto skip2

! Subroutine to show all of the data base

show_all:
!
! First, submit a query that will return all
! rows of data
!
! Start by building a string with the list
! column names separated by commas. Note the
! the first column requested is the Row Index.
! SQLite automatically generates a unique index
! value for each row. That Index is named "_id"

Columns$ = "_id," + c2$  +"," + c1$ + "," +c3$ + "," + c4$

! Note that she is asking that the rows be returned
```

```
! with the columns in the order last_name, first_name
! instead of the order in which they are in the table.
!
! Query the data base. The query will return a cursor
! that can be used to step through each of the
! returned rows.
!

sql.query Cursor, DB_Ptr, tbname$,  Columns$

! fall through into the show_results subroutine

! show_query_results subroutine

show_query_results:

Print title$

! Step the cursor to each returned row and get
! the result values. The number of result variables
! should be the the same as the number of queried
! columns. The done variable returns false (0) until
! the last row is read.

xdone = 0
do
        sql.next xdone,cursor,index$,v1$,v2$,v3$,v4$
        if !xdone then print  index$,v1$,v2$,v3$,v4$
until xdone
print " "
return

skip2:

! The first person who asks for their grade
! is, of course, Betty Ng. Ms. O'Brien formats
! a query to get just Betty's data.
!
! She will use the already formatted Columns$
! to return the Index and all the columns from
! Betty's data row.
!
! She creates a Where$ string that will select
! only Betty's Column

Where$ = "first_name = 'Betty' AND last_name = 'Ng' "
sql.query Cursor, DB_Ptr, tbname$,  Columns$, Where$

! Now go display the result

title$ = " Betty's grade"
gosub show_query_results

! When other students ask for their grade,
! Ms O'Brien will run a similar query with
! their name.
!
! Now Ms O'Brien would like a list of everyone
```

```
! who is making a grade of 90 or better.
!
! In this case, the Where$ string looks like this:

Where$ = "grade_average > '89' "
sql.query Cursor, DB_Ptr, tbname$,  Columns$, Where$

! Now go display the result

title$ = "People with grades of 90 or better."
gosub show_query_results;

!
! Bill Wilkinson was doing so poorly in the class
! that he decided to drop the class. Ms O'Brien
! now needs to remove him from the database.
!

Where$ = "first_name = 'Bill' AND last_name = 'Wilkinson' "
sql.delete DB_ptr, tbname$, Where$

! Now display the table without Bill

title$ = "Table without Bill Wilkinson"


! For this particular query, she wants the data
! returned in ascending order of the last name.

Order$ = "last_name ASC"
sql.query Cursor, DB_Ptr, tbname$,  Columns$, "", Order$
gosub show_query_results

! The Mid Term test results are in. Ms O'Brien
! needs to update the grades. She will start
! by changing Tamasin's grade. Her attendance
! record has not changed.

Where$ = "first_name = 'Tamasin' AND last_name = 'Washington' "
sql.update DB_Ptr, tbname$, c3$, "94": Where$

! Now examine Tamasin's record to insure that
! it is correct.

sql.query Cursor, DB_Ptr, tbname$,  Columns$, Where$
title$="Tamasin's new grade"
gosub show_query_results

! Finally, Ms O'Brien would like to know the
! average grade for her class.
!
! She only needs to get the grade average column

Columns$ = "grade_average"
sql.query Cursor, DB_Ptr, tbname$,  Columns$

xdone = 0
count = 0
```

```
sum = 0

do
        sql.next xdone,cursor,v$
        if !xdone then
    count = count + 1
    sum = sum + val(v$)
    endif
until xdone

Print "The class grade average is "; format$("###.##",sum/count)

!
! After we are done with an open database, we
! should close it.

sql.close DB_Ptr

end

! In the real world, Ms O'Brien made several
! small BASIC! programs to do each  of the
! tasks of Inserting, Deleting, Updating, etc.
! She used the INPUT command to get the
```

f10_graphics_object.bas

```
!! When you run this program, the Graphics
! Screen will be displayed. To exit back
! to the BASIC! text output screen, hit
! the Back key.
!
! Open graphics screen with
! background color of WHITE
!!
gr.open 255, 255, 255, 255
gr.orientation 0  % Force landscape
gr.screen w, h
whalf = w/2
hhalf = h/2
wqtr = w/4
hqtr = h/4
factor = 0.1

! Draw divider lines

!Set the draw color to black
a = 128
r = 0
g = 0
b = 0
fill = 0
gr.color a,r,g,b,fill

x1 = 0
y1 = hhalf
x2 = w
```

```
y2 = hhalf

gr.line n, x1, y1, x2, y2

x1 = whalf
y1 = 0
x2 = whalf
y2 = h

gr.line n, x1, y1, x2, y2

! Draw unfilled red circle

r = 255
g = 0
b = 0
gr.color a,r,g,b,fill

cx = wqtr
cy = hqtr
radius = hqtr - factor*hqtr

gr.circle nc, cx, cy, radius

! Label the circle

gr.color 255,0,0,0,1
gr.text.size 30
gr.text.align 2
gr.text.draw pt, cx, cy , "Circle"

! Draw a filled red oval

fill = 1
gr.color a,r,g,b,fill

left = whalf  + factor* (wqtr)
let top = factor*(hqtr)
right = w - factor*(wqtr)
bottom = hhalf -factor*(hqtr)

gr.oval no, left, top, right, bottom

! Label the oval

gr.color 255,0,0,0,1
gr.text.draw pt, whalf + wqtr, hqtr, "Oval"

! Draw a blue filled rectangle

r = 0
b = 255
g = 0
gr.color a,r,g,b,fill

left = factor*(wqtr)
let top = factor*(hqtr) + hhalf
right = whalf - factor*(wqtr)
```

```
bottom = h - factor*(hqtr)

gr.rect nr, left, top, right, bottom

! Label the rectangle

gr.color 255,0,0,0,1
gr.text.draw pt, wqtr, (3*hqtr ), "Rectangle"


! Draw a filled yellow Arc as a Pie Chart

r = 255
g = 255
b = 0
gr.color a,r,g,b,fill

left = 3*wqtr - hqtr - factor*hqtr
let top = 3*hqtr - hqtr + factor*hqtr
right = 3*wqtr + hqtr - factor*hqtr
bottom =3*hqtr + hqtr - factor*hqtr
sa = 0
ea = 240
cf = 1

gr.arc na, left, top, right, bottom, sa,ea,cf

! Label the Pie Chart

gr.color 255,0,0,0,1
gr.text.draw pt, 3*wqtr, 3*hqtr, "Filled Arc"

! Now render everything
gr.render


! Hide and show the objects
! Pause 2 seconds before starting
! Pause 950 milliseconds with each change

pause 2000

t = 950

            gr.hide nc
              gr.render
              pause t
            gr.show nc
              gr.render
      pause t
            gr.hide no
              gr.render
              pause t
            gr.show no
              gr.render
      pause t
            gr.hide nr
              gr.render
```

```
            pause t
            gr.show nr
            gr.render
    pause t
            gr.hide na
            gr.render
            pause t
            gr.show na
            gr.render
    pause t

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

end
```

f11_graphics_touch.bas

```
! This program is designed to demonstrate
! the gr.touch and gr.bitmap, commands in
! the context of a graphics application.
!
! Run the program and follow the instructions.

Print "Touch the screen. A small star will appear."
Print "Pressing UP makes the next star bigger."
Print "Pressing DOWN makes the next star smaller"
Print "Pressing GO toggles between the star and a galaxy bitmap."
Print ""
Print "When ready, Press the SPACE key."
Print "When done, Press the BACK key."

! Wait for the SPACE key to be pressed

let done = 0
do
        inkey$ key$
        if key$ = " " then let done = 1
until done

! open the Graphics Screen with a
! white background

gr.open 255, 0,0,0

! Load the bitmap object and set mode to dots

gr.bitmap.load BM_ptr, "Galaxy.png"
m = 1

! Set the color of the dot to filled Black

gr.color 255,255,255,255,1

! Set the start radius to 1
```

```
r = 1

! Run an infinite loop testing for a touch
! and looking for key presses

do

 inkey$ key$
 if key$ = "up" then r = r + 1
 if key$ = "down" then r = r - 1
 if key$ = "go"
          if m = 1 then m = 2          else  m = 1
          endif

 gr.touch flag, x , y

 if flag
  if m = 1 then gr.circle n, x, y, r else gr.bitmap.draw n, BM_ptr , x, y
  gr.render
 endif

until 0

! When you exit the graphic screen
! with the back key, you will get
! run time error at gr.touch. This
! This because the graphics mode
! is turned off when the graphics
! screen is  exited. Hit the Back
! key one more time to get back
! to the Editor.

! The Onerror will catch the error
! and not print the graphics not
! opened error message
OnError:
end
```

f12_graphics_text.bas

```
! This example shows the different effects
! available for drawing graphical text.
!
! Open Graphics Screen with a White background.

gr.open 255, 255, 255, 255
gr.orientation 0
gr.screen w,h

xleft = w/4

! Draw a Black text alignment line

gr.color 255,0,0,0,1
gr.line n, xleft,0, xleft,h

! Set the text color to Red with fill = false
```

```
gr.color 255,255, 0, 0, 0

! Set the text size to 1/25th screen height

gr.text.size w/25

! Set the text align to Left = 1

gr.text.align 1

gr.text.draw P, xleft, h/3, "Left Aligned"

! Set the text align to Center = 2

gr.text.align 2

gr.text.draw P, xleft, h/2, "Center Aligned"

! Set the text align to Right = 3

gr.text.align 3

gr.text.draw P, xleft, 2*h/3, "Right Aligned"

! Demonstrate Text Sizes and Styles
! All examples will be aligned Left = 1
! and with Blue (0,0,255) text with fill = false

gr.text.align 1
gr.color 255, 0, 0, 255, 0

! Eight block of text will be drawn
! evenly spaced

block = h/8
xleft = w/2

gr.text.size block/4
gr.text.draw P, xleft,  block/2, "Tiny text"

gr.text.size  block
gr.text.draw blink_ptr, xleft, 2*block, "Bigger text"

! normal text size is block/2
hblock = block/2
bc = 3 % Block level

gr.text.size hblock
gr.text.bold 1
gr.text.draw BTP, xleft , bc*block + hblock, "Bold, unfilled text"
bc = bc +1

! Show Bold, filled text. Change the color
! fill parameter to true

gr.color 255, 0, 0, 255, 1
```

```
gr.text.draw P, xleft, bc*block + hblock, "Bold, filled text"
bc =bc + 1

! Show Underlinded text

gr.text.bold 0
gr.text.underline 1
gr.text.draw P, xleft, bc*block + hblock, "Underlined text"
bc =bc + 1

! Show strike through text

gr.text.underline 0
gr.text.strike 1
gr.text.draw P, xleft, bc*block + hblock, "Strike through text"
bc =bc + 1

! Show skewed text

gr.text.strike 0
gr.text.skew -0.25
gr.text.draw P,xleft, bc*block + hblock, "Skewed text"

! Now render everything

gr.render

! After a one second pause,
! start to blink the big text

pause 1000

do
  gr.hide blink_ptr
    gr.render
              pause 500
  gr.show blink_ptr
    gr.render
              pause 750
until 0

! The program does not end until the
! BACK key is pressed. When BACK
! is pressed, there will be a graphics
! not open error. The onerror will
! it an end without an error.

Onerror:
END
```

f13_animations.bas

```
! This program is designed to demonstrate
! the gr.rotate , gr.modify, audio and
! popup commands in the context of a
! graphics animation
!
```

```
! Open the Graphics screen with a background
! color of White, force landscape mode and
! get the size of the screen.

gr.open 255, 255, 255, 255
gr.orientation 0 % force landscape mode
gr.screen w,h

! Set up to draw objects with the color
! as filled Red

gr.color 255, 255, 0, 0, 1

! Draw a some rotated text

gr.text.align 1      % align left
gr.text.size h/17
text_up_x = w/5  % Up moving text start x
text_down_x = w/5   % Down moving text start x

gr.rotate.start -45, text_up_x, 180
gr.text.draw p_up_text, text_up_x, 180, "Cartman loves BASIC!"
gr.rotate.end % rotate.end signals end of this rotate
gr.render  % Render what we have done so far

! Draw and rotate a bitmap object (Cartman.png)

 ! First draw a box for Cartman to land on.
 ! The rectangle will be filled due to the
 ! fill parameter of the gr.color command
 ! issued earlier.
bleft = w/4
btop = h-55
gr.rect q, bleft, btop, bleft+68, h

! Load an image as a bitmap positioned
! on the box

! Start by loading the image bitmap
! from the Cartman.png from the directory
! "/sdcard/rfo-basic/data/"

gr.bitmap.load p_bm_obj, "Cartman.png"
! Cartman is 48 x 48 Pixels

dim p_bitmap[10] % The animation will have ten steps

! Draw Cartman on the box
! the box is 68 pixels wide and
! 55 pixels hight

index = 1   % This is the Cartman on box index
gr.bitmap.draw p_bitmap[index], p_bm_obj, bleft + 10, btop - 48
gr.render               % Render it now
pause 500               % Pause for 1/2 second
gr.hide p_bitmap[index]  % and then hide it

index = 2  % Index 2 is start of jumped, rotating Cartman
```

```
! Rotate Cartman through 9 positions

for angle = 0 to 360 step 45  % For each angle

! draw one rotation of Cartman
   gr.rotate.start angle, bleft + 10 + 24, btop-124
   gr.bitmap.draw p_bitmap[index], p_bm_obj,bleft + 10, btop - 148
   gr.rotate.end
   gr.render           % Render the rotated object

   gr.hide p_bitmap[index]

   index = index + 1

next angle  % step through all 9 postions

gr.show p_bitmap[1] % Leave Cartman standing on box

audio.load whee, "whee.mp3"
audio.load boing, "boing.mp3"

! All objects are now drawn
! Now animate them

loop:   % Start of animation loop

for index = 1 to 10 % 10 animation steps

     gr.show p_bitmap[index] % Show Cartman

! If Cartman is about jump, start the whee sound
     if index = 2 then audio.play whee

! Move the up text up by 10 pixels, If text
! goes off screen top, move to bottom

   gr.modify p_up_text, "x", text_up_x
      text_up_x = text_up_x + w/20
            if text_up_x > w/2 then text_up_x = -w/2
!!
! Move the down text down by 10 pixels. If
!  text goes off screen bottom move to top

   gr.modify p_down_text, "x", text_down_x
      text_down_x = text_down_x - 10
            if text_down_x < 10 then text_down_x = 750
!!
   gr.render % Render this animation step

 ! If Cartman just landed on the box, play
 ! boing sound and pause for 1/2 second

if index = 1 then
    audio.stop
     audio.play boing
     pause 300
     audio.stop
```

```
endif

! End of one animation step
! hide Cartman
pause 100 % To smooth animation
gr.hide p_bitmap[index]

next index    % do next animation step

% 10 animation steps done

audio.stop    % stop Cartman's whee sound
goto loop  % do the animations forever


 OnError:
 end
```

f14_compass.bas

```
! This program is designed to demonstrate
! the gr.touch and gr.bitmap, commands in
! the context of a graphics application.
!
! Run the program and follow the instructions.

Print "Touch the screen. A small star will appear."
Print "Pressing UP makes the next star bigger."
Print "Pressing DOWN makes the next star smaller"
Print "Pressing GO toggles between the star and a galaxy bitmap."
Print ""
Print "When ready, Press the SPACE key."
Print "When done, Press the BACK key."

! Wait for the SPACE key to be pressed

let done = 0
do
        inkey$ key$
        if key$ = " " then let done = 1
until done

! open the Graphics Screen with a
! white background

gr.open 255, 0,0,0

! Load the bitmap object and set mode to dots

gr.bitmap.load BM_ptr, "Galaxy.png"
m = 1

! Set the color of the dot to filled Black

gr.color 255,255,255,255,1

! Set the start radius to 1
```

```
r = 1

! Run an infinite loop testing for a touch
! and looking for key presses

do

 inkey$ key$
 if key$ = "up" then r = r + 1
 if key$ = "down" then r = r - 1
 if key$ = "go"
          if m = 1 then m = 2          else  m = 1
          endif

 gr.touch flag, x , y

 if flag
  if m = 1 then gr.circle n, x, y, r else gr.bitmap.draw n, BM_ptr , x, y
  gr.render
 endif

until 0

! When you exit the graphic screen
! with the back key, you will get
! run time error at gr.touch. This
! This because the graphics mode
! is turned off when the graphics
! screen is  exited. Hit the Back
! key one more time to get back
! to the Editor.

! The Onerror will catch the error
! and not print the graphics not
! opened error message
OnError:
end
```

f15_gps.bas

```
! Demonstrates the GPS commands

gps.open
gr.open 255, 255, 255, 255
gr.orientation 0
gr.screen w, h

sp = h/7
pad = 0.25 * sp
x = 20

 loop:

gr.text.size sp - 2*pad
gr.color 255,0,0,0,1

y  = 0*sp + sp - pad
```

```
gps.provider provider$
gr.text.draw  p, x , y , "Provider: " +  provider$

y  = 1*sp + sp - pad
gps.accuracy accuracy
gr.text.draw p, x,y, "Accuracy: " + format$("##", accuracy)

y  = 2*sp + sp - pad
gps.latitude latitude
gr.text.draw p, x,y, "Latitude: " + format$("##%.#####", latitude)

y  = 3*sp + sp - pad
gps.longitude longitude
gr.text.draw p, x, y, "Longitude: " + format$("##%.#####", longitude)

 y  = 4*sp + sp - pad
 gps.altitude altitude
gr.text.draw p, x, y, "Altitude: " + format$("#####", altitude)

y  = 5*sp + sp - pad
gps.bearing bearing
gr.text.draw p, x, y, "Bearing: " + format$("##%.##", bearing)

y  = 6*sp + sp - pad
gps.speed speed
gr.text.draw p, x,y, "Speed: " + format$("##%.##", speed)

gr.render
pause 5000
gr.cls

goto loop

onerror:
end
```

f16_select.bas

```
!!

Demonstrates the Select Command

!!

! Load an array with the names of the months

array.load months$[], "January", "February"~
"March", "April", "May", "July", "August"~
"September", "October", "November", "December"

! Set the Popup Message

msg$ ="Select the month of your birth."

! Shows the list and waits for the user
! to make the selection.

select month, months$[], msg$
```

```
! if the selection was zero, the user hit the
! BACK key

if month = 0
  print "You are unborn"
else
  print "You were born in ";
  print months$[month]
endif
```

f17_gravity.bas

```
!!
This program demonstrates the Accelerometer Sensors.

It displays the magnitude and direction of the
accelerometer data in the X and Y axis as Red
and Green lines.

The magnitude and direction of the X and Y
vectors are then added to generate the
gravity vector blue line.

The blue line will always point to the center
of the Earth.

If you hold the device horizonatally the three
displayed vectors will disappear.

!!

! Open the acclerometer sensor
sensors.open 1

! Open Graphics
gr.open 255,0,0,0
gr.orientation 0

! Get screen size paramters
gr.screen w,h

! Calculate screen center x, y
cx = w/2
cy = h/2

! The main program loop
do

!Read the acclerometer
sensors.read 1,y,x,z

!Amplify the magnitudes
x = x* (cy/12)
y = y*(cy/12)
```

```
! Draw the X Axis in Red
gr.color 255,255,0,0,1
gr.line q, cx, cy, cx, cy+ y

! Daw the Y Axis in Green
gr.color 255,0,255,0,1
gr.line r, cx, cy, cx+ x, cy

! Calculate the gravity vector
m = sqr(x*x + y*y)
if m=0 then m = 0.0001
angle = todegrees(acos(x/m))
if y < 0 then angle = - angle

! Draw the gravity vector
gr.color 255,0,0,255,1
gr.rotate.start angle, cx, cy
gr.line s, cx, cy, c x+m, cy
gr.rotate.end

! Now show what we have drawn
gr.render

! Pause for a moment
pause 5

! Clear the screen and repeat
gr.cls

until 0

 ! If the user hits BACK, he will not see
 ! a Graphics not opened message
 ! because OnError will intercept

OnError:
end
```

f18_time.bas

```
! TIME Command
!
! TIME Year$, Month$, Day$, Hour$, Minute$, Second$
!
! All the paramters are returned by the TIME command.
!
! Year$: The four digit year
! Month$: The two digit month (1 to 12)
! Day$: The two digit day of the month (1 to 31)
! Hour$: The two digit hour (0 to 23)
! Minute$: The two digit minute (0 to 59)
! Second$: The two digit second (0 to 59)
!

DIM m$[12]
m$[1] = "January"
```

```
m$[2] = "February"
m$[3] = "March"
m$[4] = "April"
m$[5] = "May"
m$[6] = "June"
m$[7] = "July"
m$[8] = "August"
m$[9] = "September"
m$[10] = "October"
m$[11] = "November"
m$[12] = "December"

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

Print "Today's Date Is:"
Print m$[val(Month$)]; " "; Day$; ", "; Year$
Print ""

AMPM $= " AM"
Hour = val(Hour$)

IF Hour = 0 THEN
  theHour = 12
  ELSE
    IF Hour = 12
            AMPM$ = " PM"
                  theHour = 12
                  ELSE
                    IF Hour < 13 THEN
                      theHour = Hour
                            ELSE
                              AMPM$ = " PM"
                                    theHour = MOD(Hour, 12)
                          ENDIF
    ENDIF
          ENDIF

Print "The Time Is:"
Print format$("##",theHour); ":"; Minute$; AMPM$
```

f19_towers_of_hanoi.bas

```
!!
Solves the Tower of Hanoi puzzle using functions
and recursion.

The puzzle has disks stacked on one of three pegs.
The object is to move all the disks from that peg
to another peg. The rule is that no disk may ever
be on top of a smaller disk

!!

!****  Hanoi Algorithm ****

fn.def hanoi(disk,dest,source,other, peg[],  parms[])
```

```
! the parms peg[], and parms[] are passed through
! for drawing and are not used the algoritim

if disk > 0
n = hanoi(disk-1,other,source,dest, peg[], parms[])
n = move(source, dest, peg[], parms[])
n = hanoi(disk-1,dest,other,source, peg[], parms[])
fn.rtn 0
endif
fn.rtn 0
fn.end

!**** dleft ****
! Calculate the x coordinate of left side of a disk given
! the disk number and margin

fn.def dleft(disk, w, peg, margin)
c = w/4 - disk * margin
q = c/2
p = peg*(w/4)
fn.rtn p-q
fn.end

!**** dright ****
! Calculate the x coordinate of the right side of a disk
! given the disk number and margin

fn.def dright(disk, w, peg, margin)
c = w/4 - disk * margin
q = c/2
p = peg*(w/4)
fn.rtn p + q
fn.end

!**** dtop ****
! Calculate the y coordinate of the top of a disk
! given the disk level on the peg and the
! space between the disks

fn.def dtop(level, height, space)
b = height - (level * 2 * space)
t = b - space
fn.rtn t
fn.end

!**** dbottom ****
! Calculate the y coordinate of the bottom  of a disk
! given the disk level on the peg and the
! space between the disks

fn.def dbottom(level, height, space)
b = level * 2 * space
fn.rtn height - b
fn.end

!**** move ****

! Move one disk
```

```
fn.def move( from, to, peg[,], parms[,])
i = 1
! find the first non-empty level on the from peg

while peg[from,i] <> 0
i = i +1
repeat

! back up to the last non-empty peg
i = i -1

! Get the disk number
disk = peg[from,i]

! Remove the disk from the peg
peg[from,i] = 0

i = 1
! find the first non-empty level on the to peg

while peg[to, i] <>0
i = i +1
 repeat

! put the disk in that first empty level
peg[to, i] = disk

! now go draw the pegs and disks
n = draw_it( peg[,], parms[,] )

fn.rtn 0
fn.end

!**** drawit ****
! Draw the pegs and disks

fn.def draw_it( peg[,] , parms[,])

gr.cls

! Get the values out of parms[,]
h = parms[1]
w = parms[2]
space = parms[3]
margin = parms[4]

! Color the peg red
gr.color 255,255,0,0,1

! Draw the pegs
gr.line f, w/4, h, w/4, h/3
gr.line f, w/2, h, w/2, h/3
gr.line f, 3*w/4, h, 3*w/4, h/3

! Color the disks blue
gr.color 255,0,0,255,1

! For each peg
```

```
for thePeg = 1 to 3

!Draw each disk current on the peg
theLevel = 1
while peg[thePeg, theLevel] <> 0
disk = peg[ thePeg, theLevel]

! get the coordinates of the disk
! at this peg and this level
left = dleft(disk, w, thePeg, margin)
right = dright(disk,w,thePeg,margin)
xtop = dtop(theLevel, h, space)
bottom = dbottom(theLevel, h, space)

! Draw one disk
gr.rect f, left, xtop, right, bottom
theLevel = theLevel + 1

repeat
next thePeg

! Draw everything
gr.render
pause 500

fn.rtn 0
fn.end

!**** Main Program ****

! Number of disks
disk_count = 4

! Define the pegs and fill the first peg with the disks
! Disk 1 is the the largest disk

! Dimension array such there will always be a disk
! zero at the top of the peg

dim peg[3,disk_count+1]

! Fill peg 1
for i = 1 to disk_count
peg[1,i] = i
next i


gr.open 255,255,255,255
gr.orientation 0
gr.screen width, height

! Largest disk is 1/4 screen width minus
! margin. Smaller disks are reduced
! by margin

margin = 20

! space between disks and height
```

```
! of the disks

space = height/20

dim parms[4]
parms[1] = height
parms[2] = width
parms[3] = space
parms[4] = margin

! Draw the initial pegs and disks
n = draw_it(peg[], parms[])

! Solve the puzzle
n =hanoi(disk_count, 3, 1, 2, peg[], parms[])

! Done

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


end
```

f20_switch.bas

```
! This program demonstrates the
! SWITCH commands

Print "How many days in a given month?"
Print  " "
input "Enter Month Name", aname$
name$ = lower$(aname$)
num_days = 0

sw.begin name$
sw.case "january"
sw.case "march"
sw.case "may"
sw.case "july"
sw.case "august"
sw.case "october"
sw.case "december"
num_days = 31
sw.break

sw.case "april"
sw.case "june"
sw.case "september"
sw.case "november"
num_days = 30
sw.break

sw.case "february"

TIME Year$, Month$, Day$, Hour$, Minute$, Second$
year = val(Year$)

if ( (mod(year,4) =0) & (mod(year,100) <>0) |  (mod(year, 400) = 0) then
  numDays - 29
 else
  num_days = 28
 endif
sw.break

sw.default
print "Invalid month"

sw.end

if num_days <>0
print "The number of days in " + aname$ + " is:" + format$("##",num_days)
endif

END
```

F21_sos.bas

```
!!
This program demonstrates the vibrate
command by vibrating out SOS in
```

Morse Code.
!!

! Set the variables

xdot = 200 % Length of a Morse Code "dot" in milliseconds
dash = 500 % Length of a Morse Code "dash" in milliseconds
sg = 200 %Length of Gap Between dots/dashes
mg = 500 %Length of Gap Between Letters
lg = 1000 % Length of Gap Between Words

! Load the Pattern[] Array with the SOS
! no initial pause
! dot, dot, dot
! dash, dash, dash
! dot, dot, dot

array.load Pattern[], 0~
xdot, sg, xdot, sg, xdot, mg~
dash,sg,dash,sg,dash,mg~
xdot, sg, xdot, sg, xdot

vibrate Pattern[], -1
pause 5500
end

f22_benchmark.bas

!!
This is a version of a Whestone Benchmark
written for QBasic and adapted for BASIC! The
calculation of MWIPS and MFLOPS have
been removed.

How fast is you Android device?

Run this test and post the results on the BASIC!
forum at:
http://rfobasic.freeforums.org/

!!
```
Rem
Rem  Document:        Whets.bas
Rem  File Group:      Classic Benchmarks
Rem  Creation Date:   9 December 1996
Rem  Revision Date:
Rem
Rem  Title:           Whetstone Benchmark for QBasic
Rem  Keywords:        WHETSTONE BENCHMARK PERFORMANCE MIPS
Rem                   MWIPS MFLOPS
Rem
Rem  Abstract:        QBasic version of Whetstone one of the
Rem                   Classic Numeric Benchmarks with example
Rem                   results for PCs.
Rem
Rem  Contributor:     Roy Longbottom 101323.2241@compuserve.com
Rem                      or     Roy_Longbottom@compuserve.com
Rem
```

```
Rem************************************************************

Rem     QBasic Whetstone benchmark Single Precision
Rem
Rem     Original concept        Brian Wichmann  NPL     1960Rems
Rem     Original author         Harold Curnow   CCTA    1972
Rem     Self timing versions    Roy Longbottom  CCTA    1978/87
Rem     Optimisation control    Bangor University       1987
Rem     PC versions             Roy Longbottom          1996
Rem
Rem************************************************************
Rem
Rem This version is a greatly simplified version of the full
Rem Whetstone benchmark. The results of this benchmark are
Rem only relative to the results of other devices running
Rem this program.
Rem
Rem************************************************************

! Returns the time in seconds resolved to the hour
! If run on the cusp of the hour, will produce bad data

 fn.def xtime()
 time Y$, M$, D$, H$, M$, S$
 fn.rtn val(H$) * 3600 + val(M$)*60 + val(S$)
 fn.end

   DIM ex[4]
   DIM results[8]
   DIM ztime[8]
   DIM heading$[8]
   DIM ops[8]
   DIM flops[8]

   icount = 10
   calibrate = 0
   ixtra = 1
   ix100 = 1

   Rem Passes to average
   Passes = 10

   CLS

   Print ""
   PRINT "Whetstone Benchmark modified for BASIC!"
   PRINT ""
   Print "Performing ";Passes;" Passes"

   mTimeUsed = 0
   Check = 0
   For Pass = 1 to Passes
      GOSUB Whetstones
   Next Pass

   PRINT ""
   PRINT "Total Time: "; mTimeUsed; " secs"
   PRINT ""
```

```
   PRINT "Average Pass Time: "; mTimeUsed / Passes; " secs"
   PRINT ""

   IF Check = 0 THEN
      PRINT "Calculation Error Detected"
   Else
      Print "Calculations Correct"
   Endif

   End
```

Whetstones:

```
   Rem   INITIALISE CONSTANTS

   t = 0.49999975
   t0 = t
   t1 = 0.50000025
   t2 = 2

   n1 = 12 * ix100
   n2 = 14 * ix100
   n3 = 345 * ix100
   n4 = 210 * ix100
   n5 = 32 * ix100
   n6 = 899 * ix100
   n7 = 616 * ix100
   n8 = 93 * ix100
   n1mult = 10

   Print ""
   Print "Pass: ";Pass;" of ";Passes
   Print ""

   Rem MODULE 1 - ARRAY ELEMENTS
  let  stime = xTime()
 let   ex[1] = 1
 let   ex[2] = -1
  let  ex[3] = -1
  let  ex[4] = -1
   FOR ix = 1 TO ixtra
      FOR i = 1 TO n1 * n1mult
    let       ex[1] = (ex[1] + ex[2] + ex[3] - ex[4]) * t
      let     ex[2] = (ex[1] + ex[2] - ex[3]+ ex[3]) * t
       let    ex[3] = (ex[1] - ex[2] + ex[3] + ex[4]) * t
       let    ex[4]= (-ex[1] + ex[2] + ex[3] + ex[4]) * t
      NEXT i
   let     t = 1.0 - t
   NEXT ix
  let  t = t0
  let  checsum = ex[4]
 let   rtime = (xTime() - stime)
   smflops = n1 * 16
   title$ = "N1 floating point"
   atype = 1
   section = 1
```

Rem N1 * 16 floating point calculations

```
   GOSUB Pout

   Rem MODULE 2 - ARRAY AS PARAMETER
   stime = xTime()
   FOR ix = 1 TO ixtra
      FOR i = 1 TO n2

         GOSUB PA

      NEXT i
 let      t = 1.0 -t
   NEXT ix
 let   t = t0
  let  checsum = ex[4]
  let  rtime = xTime() - stime
   smflops = n2 * 96
   title$ = "N2 floating point"
   atype = 1
   section = 2
Rem  N2 * 96 floating point calculations
   GOSUB Pout

   Rem MODULE 3 - CONDITIONAL JUMPS
  let   stime = xTime()
  let  j = 1
   FOR ix = 1 TO ixtra
      FOR i = 1 TO n3
         IF j <> 1 THEN
           j = 3
         ELSE
           j = 2
         Endif
         IF j <= 2 THEN
           j = 1
         ELSE
           j = 0
         Endif
         IF j >= 1 THEN
           j = 0
         ELSE
           j = 1
         Endif
      NEXT i
   NEXT ix
  let  checsum = j
  let  rtime = xTime() - stime
   smflops = n3 * 3
   title$ = "N3 if then else"
   atype = 2
   section = 3

Rem  N3 * 3 IF THEN ELSE
   GOSUB Pout

   Rem MODULE 4 - INTEGER ARITHMETIC
  let   stime = xTime()
  let    j = 1
  let    k = 2
```

```
  let   l = 3
    FOR ix = 1 TO ixtra
      FOR i = 1 TO n4
     let      j = j * (k - j) * (l - k)
      let     k = l * k - (l - j) * k
     let      l = (l - k) * (k + j)
     let      ex[l - 1] = j + k + l
      let     ex[k - 1] = j * k * l
      NEXT i
    NEXT ix
  let   checsum = ex[2] + ex[1]
  let   rtime = xTime() - stime
    smflops = n4 * 15
    title$ = "N4 fixed point"
    atype = 2
    section = 4

Rem  N4 * 15 fixed point operations
   GOSUB Pout

    Rem MODULE 5 - TRIG. FUNCTIONS
  let  stime = xTime()
  let    X = 0.5
  let    Y = 0.5
    FOR ix = 1 TO ixtra
      FOR i = 1 TO n5
  let       X = t * ATAN(t2 * SIN(X) * COS(X) / (COS(X + Y) + COS(X - Y) - 1))
    let     Y = t * ATAN(t2 * SIN(Y) * COS(Y) / (COS(X + Y) + COS(X - Y) - 1))
      NEXT i
  let     t = 1.0 - t
    NEXT ix
   let  t = t0
  let   checsum = Y
  let   rtime = xTime() - stime
    smflops = n5 * 26
    title$ = "N5 sin,cos etc."
    atype = 2
    section = 5

Rem  N5 * 26 function calls and floating point operations
   GOSUB Pout

    Rem MODULE 6 - PROCEDURE CALLS
  let   stime = xTime()
  let   X = 1
   let  Y = 1
  let   Z = 1
    FOR ix = 1 TO ixtra
      FOR i = 1 TO n6

         GOSUB P3

      NEXT i
    NEXT ix
 let    checsum = Z
  let   rtime = xTime() - stime
  let   smflops = n6 * 6
    title$ = "N6 floating point"
```

```
    atype = 1
    section = 6

Rem N6 * 6 floating point operations
   GOSUB Pout

   Rem MODULE 7 - ARRAY REFERENCES
  let  stime = xTime()
 let   j = 1
 let   k = 2
  let  l = 3
 let   ex[1] = 1
  let  ex[2] = 2
  let  ex[3] = 3
   FOR ix = 1 TO ixtra
      FOR i = 1 TO n7

         GOSUB PO

      NEXT i
   NEXT ix
  let  checsum = ex[3]
 let   rtime = xTime() - stime
   smflops = n7 * 3
   title$ = "N7 assigns"
   atype = 2
   section = 7

Rem N7 * 3 assignments
   GOSUB Pout

   Rem MODULE 8 - STANDARD FUNCTIONS
 let   stime = xTime()
 let   X = 0.75
   FOR ix = 1 TO ixtra
      FOR i = 1 TO n8

  let         X = SQR(LOG(X) / t1)

      NEXT i
   NEXT ix
 let   checsum = X
  let  rtime = xTime() - stime
   smflops = n8 * 4
   title$ = "N8 exp,sqrt etc."
   atype = 2
   section = 8

Rem N8 * 4 function calls and floating point operations
   GOSUB Pout

   RETURN
Rem END OF MAIN ROUTINE



PA:
```

```
    Rem PROCEDURE PA
    j = 0
    DO
  let         ex[1] = (ex[1] + ex[2] + ex[3] - ex[4]) * t
   let       ex[2] = (ex[1] + ex[2] - ex[3]+ ex[3]) * t
    let      ex[3] = (ex[1] - ex[2] + ex[3] + ex[4]) * t
     let     ex[4]= (-ex[1] + ex[2] + ex[3] + ex[4]) * t/2
    let   j = j + 1
   until j = 6
   RETURN

PO:
    Rem PROCEDURE P0
  let   ex[j] = ex[k]
  let   ex[k] = ex[l]
  let   ex[l] = ex[j]
   RETURN

P3:
   Rem PROCEDURE P3
  let   X = Y
  let   Y = Z
 let    X = t * (X + Y)
 let    Y = t1 * (X + Y)
 let    Z = (X + Y) / t2
   RETURN

Pout:

   Check = Check + checsum
   ztime[section] = rtime
   heading$[section] = title$
   mTimeUsed = mTimeUsed + rtime


   IF calibrate = 1 THEN
      results[section] = checsum
      PRINT "#";
   Endif

   IF calibrate = 0 THEN
      PRINT heading$[section];

      PRINT "  ";
      PRINT ztime[section]; " sec"

   Endif
   RETURN
```

F23_breakout.bas

```
!!
This game of Breakout is intended
as an example of collision
detection in Graphics mode.

 ***** Instructions *****
```

The ball will only move while
touching the screen. Lift your
finger from the screen and the
ball will stop. Touch the screen
again and the ball will move.

You can put some "English" on
the ball depending on the speed
and direction of the paddle strike

After the ball stops flashing at
game end, you can start a new
game. Just tap the screen.

The score is the number of bricks
remaining after breaking out. The
theoretical maximum score is 21.

You can change the speed of the
action by changing the "friction"
value just below this comment.
Larger friction values result
in slower action.

!!

```
friction = 15
gr.open 255, 0,0,0
gr.orientation 0
gr.screen sw, xsh
sh = xsh - xsh/10
space = 2
last_score = 0
high_score = 0
lives = 0

gosub instructions

! Game restarts here

restart:
english = 0
lives = lives + 1

! Draw bounding lines
! Bounding lines are filled rectangles

line_width = ceil(sh/200)

! Make top line black
! It will not be seen
! but will report collisions

gr.color 255, 0, 0, 0, 1
! Top Line
gr.rect TL, 0, 0, sw, line_width

! Make what follows white
```

```
gr.color 255, 255, 255, 255, 1

! Display the score information

gr.text.size xsh/10 - 6
gr.text.align 1
gr.text.draw text, 0, xsh - 8, " Lives:" + format$("%%",lives) + "  Last Score:" + format$("%%",last_score) + "
High Score:" + format$("%%",high_score)

! Left line
gr.rect  LL, 0, 0, line_width, sh
! Right Line
gr.rect RL, sw - 2*line_width, 0, sw, sh
! Bottom Line
gr.rect  BL, 0, sh - line_width, sw, sh

! Draw Paddle

paddle_count = 4
paddle_height = sh/20
pb = sh - line_width - space
pt = pb - paddle_height
pl = line_width + space
pr = pl + sw/paddle_count - 2*line_width - 2*space
paddle_width = pr - pl
paddle_left_limit = pl
paddle_right_limit = sw - paddle_width - line_width - space

! Adjust paddle postion to center

pl = sw/2 - paddle_width/2
pr = pl + paddle_width
gr.rect Paddle, pl, pt, pr, pb

! Draw the Ball

! If this is a game restart
! position the ball where
! the user touched the screen

 ball_x = rnd(1)*sw
 ball_y = sh/2

! The _xx values are the
! ball movement increments

ball_xx = -15
ball_yy = 15

ball_radius = paddle_height
gr.circle Ball, ball_x, ball_y, ball_radius

! Draw the target bricks
! Each row of bricks will have
! different colors

T_count = 8
```

```
T_width = sw/T_count
T_width = T_width - 2*line_width + space
T_start_left = 2*line_width + space
T_height = Paddle_height
T_start_top = T_height

! Top Row

 ttl = T_start_left
 tt = T_start_top
 tr = ttl + T_width
 tb = tt + T_height
 gr.color 255,255,0,0,1
 dim T1[T_count]
 for i = 1 to T_count
  gr.rect T1[i], ttl, tt, tr, tb
  ttl = tr + space
  tr = ttl + T_width
 next i

! Middle Row

tt = tt + T_height + space
tb = tt + T_height
ttl = T_start_left
tr = ttl + T_width
dim t2[T_count]
gr.color 255,0,255,0,1
for i = 1 to T_count
 gr.rect T2[i], ttl, tt, tr, tb
 ttl = tr + space
 tr = ttl + T_width
next i

 ! Bottom Row

tt = tt + T_height + space
tb = tt + T_height
ttl = T_start_left
tr = ttl + T_width
dim T3[T_count]
gr.color 255,0,0,255,1
for i = 1 to T_count
 gr.rect T3[i], ttl, tt, tr, tb
 ttl = tr + space
 tr = ttl + T_width
next i

! The lower limit is where
! we start looking for
! collisions with bricks

 T_lower_limit = tb + T_height

! Finally, render the screen

gr.render
```

```
! Some final variables to set

gr.color 255,255,255,255,1
game_end = 0
hits = 0

! ***** Main Game Loop *****

do

gr.touch touched, tx1, ty1
while touched & (game_end = 0)

! Move the paddle

  gr.touch touched, tx2, ty2
  delta_tx = tx2 -tx1
  tx1 = tx2
  npl = pl + delta_tx
  if (npl <= paddle_left_limit) | npl >= (paddle_right_limit) then npl = pl
  gr.modify Paddle, "left", npl
  gr.modify Paddle, "right", npl + paddle_width
  pl = npl

! Move the ball

  ball_x = ball_x + ball_xx + english
  if ball_x > sw then ball_x = sw - ball_radius
  if ball_x <  0 then ball_x = ball_radius
  ball_y = ball_y + ball_yy
  gr.hide Ball
  gr.circle Ball, ball_x, ball_y, ball_radius

! Check for collision with Paddle

  if gr_collision(Ball, Paddle)
   ball_yy = -ball_yy
   if abs(delta_tx) >10
    english = delta_tx/2
    endif
   endif

! Check for collisions with the
! boundry lines

  if gr_collision(Ball, LL)
    ball_xx = abs(ball_xx)
    english = 0
  endif

  if gr_collision(Ball, RL)
    ball_xx = - abs(ball_xx)
    english = 0
   endif

  if gr_collision(Ball, BL) then game_end = 1
  if gr_collision(Ball, TL) then game_end = 2
```

```
! Check for collisions with the bricks
! Checks will only be made if ball
! is past the threshold

  if (game_end = 0) & (ball_yy < 0) & (ball_y < T_lower_limit)
   hit = 0

! Bottom row

   for i = 1 to T_count
            if gr_collision(Ball, T3[i])
             gr.hide T3[i]
             ball_yy = -ball_yy
             i = T_count + 1
             hit = 1
             hits = hits + 1
             english = 0
            endif
   next i

! Middle Row
! Will only be checked if no hit
! on bottom row

            if (hit = 0)
     for i = 1 to T_count
              if gr_collision(Ball, T2[i])
               gr.hide T2[i]
               ball_yy = -ball_yy
               i = T_count + 1
               hit = 2
               hits = hits + 1
               english = 0
              endif
             next i
   endif

! Top row
! Will only be checked if no hit
! on bottom two rows

            if (hit =  0)
     for i = 1 to T_count
              if gr_collision(Ball, T1[i])
               gr.hide T1[i]
               ball_yy = -ball_yy
               i = T_count + 1
               hit = 2
               hits = hits + 1
               english = 0
              endif
             next i
            endif

  endif

! Show where we are at and
! then pause by the friction
```

```
! amount.

  gr.render
  pause friction

repeat
until game_end

! End of game action
! Keep score

 if game_end = 1 then hits = 24
 last_score = 24 - hits
 if last_score > high_score then high_score = last_score

gr.hide text
gr.text.draw z, 0, xsh - 8, " Lives:" + format$("%%",lives) + "  Last Score:" + format$("%%",last_score) + "  High
Score:" + format$("%%",high_score)

! Flash the ball

  for i = 1 to 3
   gr.hide Ball
   gr.render
   pause 250
   gr.show Ball
   gr.render
   pause 250
  next i

  gr.text.draw z, sw/12, sh/2, "Tap the screen to start new game."
  gr.render

! Wait for the new game
! touch and release signal
! The touch point will be
! the ball start location
! for the new game

do
 gr.touch touched,nx,ny
until touched
do
 gr.touch t,b,b
until !t

! Prep for new game

gr.cls
undim T1[]
undim T2[]
undim T3[]
goto restart

! Catch graphics not
! open error message
! and end
```

```
!onerror:
end

instructions:

dim line$[20]
i = 1
line$[i]= "              *** INSTRUCTIONS *** "
i = i + 1
line$[i] = " "
i = i + 1
line$[i]= "The ball will move only while touching the screen."
i = i+1
line$[i] = " "
i = i + 1
line$[i]= "The speed and direction of the paddle strike affects the"
i = i+1
line$[i]= "movment of the ball."
i = i+1
line$[i] = " "
i = i + 1
line$[i]= "The game score is the number of bricks remaining after"
i = i+1
line$[i]= "breaking out."
i = i + 1
line$[i] = " "
i = i + 1
line$[i]= "Start the game by tapping the screen now."
line_count = i

it_size = sh/15
it_space = sh/45
gr.color 255, 255, 255, 255, 1
gr.text.align 1
gr.text.size it_size
y = it_space + it_size
x = 0

for k = 1 to line_count
 gr.text.draw z, x , y,line$[k]
 y = y + it_space + it_size
next i

gr.render

do
 gr.touch touched,nx,ny
until touched
do
 gr.touch t,b,b
until !t

gr.cls
return
```

f24_newdl.bas

```
!!
```

```
This program demonstrates the use
of the gr.NewDL to change the display list
and thus change the Z order
of objects.

Coincidentally, it demonstrates
the some of BASIC!s array manipulation
capabilities
!!

gr.open 255,255,255,255
gr.orientation 0
gr.screen width, height
count = 8
delay = 3000

s =( width/(count))/4
center = height/2
@top = center - 5*s

dim objs[count]

!!
Draw count rectangles
4*s high
4*s wide with a 1*s overlap

Each rectangle will have
a random color

The first rect starts at
3*s
!!

left = 3*s

for i = 1 to count
 rr = rnd() * 255
 gg = rnd() * 255
 bb = rnd() * 255
 gr.color 240, rr, gg, bb, 1
 gr.rect objs[i], left, @top, left + 4*s, @top + 4*s
 left = left + 3*s
 @top = @top + s
next i

! Display loop

do
 !draw in original order

 gr.render
 pause delay

 ! draw in reverse order

 array.reverse objs[]
 gr.newdl objs[]
```

```
gr.render
pause delay

! Do four cycles with
! random shuffling

for j = 1 to 4
 array.shuffle objs[]
 gr.newdl objs[]
 gr.render
 pause delay
next j

! Restore to the original order

array.sort objs[]
gr.newdl objs[]

until 0
```

f25_dir.bas

```
!!
This program can be used to
explore the file system on an
Android device.

Any filename with a (d) after
it is a directory. Tap on that
entry to open that directory.

Tap on the top entry, "..", to
move up one directory level.

Press the BACK key to exit.

This program will list all the
way up to the file system root.

The initial directory shown is
"/sdcard/rfo-basic/data/"

!!

! Lists a directory. Gets and
! returns the user's selection

path$ = ""
loop:

! Get the listing from
! the path directory
! and sort it

array.delete d1$[]
file.dir path$, d1$[]
array.length length, d1$[]
```

```
! Copy the file list
! adding the top ".."
! entry

array.delete d2$[]
dim d2$[length+1]
d2$[1] = ".."
for i = 1 to length
 d2$[i + 1] = d1$[i]
next i

! Present the list
! and get the user's
! choice

select s, d2$[], ""
if s = 0 then END

! If top entry, "..", not
! selected then append
! the selected directory
! name to the path

if s>1
 n = is_in("(d)", d2$[s])
 if n = 0
  goto loop
  endif
 dname$ = left$(d2$[s],n-1)
 path$=path$+dname$+"/"
 goto loop
endif

! If s = 1 then must back
! one level

! if at start path then
! back up one level

 if path$ = ""
  path$ = "../"
  goto loop
 endif

! Not at start path
! split the path by
! the "/" chars

 array.delete p$[]
 split p$[], path$, "/"
 array.length length, p$[]

! If the last entry is
! ".." then add "../"
! to back up one level

 if p$[length] = ".."
  path$ = path$ + "../"
```

```
  goto loop
  endif

! Last entry is not ".."
! so must delete the
! last directory from
! the path

! If only one entry
! then path is back
! to the base path

 if length = 1
  path$ = ""
  goto loop
  endif

! Reconstruct path without
! the last directory

 path$ = ""
 for i = 1 to length - 1
  path$ = path$ + p$[i] + "/"
 next i

 goto loop
```

f26_array_copy.bas

```
! This program demonstrates
! the array.copy command

! This function prints a numeric
! Array

fn.def array_print(a[],msg$)
 print msg$
 array.length l,a[]
 for i = 1 to l
  print a[i]
 next i
 fn.rtn 0
fn.end

array.load n1[],1,2,3,4,5
n=array_print (n1[],"Original array")

array.copy n1[],n2[]
n=array_print (n2[], "Straight copy")
```

```
array.copy  n1[],n3[2]
n=array_print (n3[], "2 extras at end")

array.copy n1[],n4[-2]
n=array_print (n4[], "2 extras at front")

array.copy n1[2,3], n5[]
n=array_print (n5[], "trim to middle three"

array.copy n1[1,2],n6[-1]
n=array_print (n6[], "trim to first 2 and add 1 at start")

! This function prints a String
! Array

fn.def array_print$(a$[],msg$)
 print msg$
 array.length l,a$[]
 for i = 1 to l
  if a$[i] = "" then print "(empty)" else print a$[i]
 next i
fn.rtn ""
fn.end

array.load str$[],"a","b","c","d","e"
n$=array_print$ (str$[],"Original array")

array.copy str$[],n2$[]
n$=array_print$ (n2$[], "Straight copy")

array.copy  str$[],n3$[2]
n$=array_print$(n3$[], "2 extra at end")

array.copy str$[],n4$[-2]
n$=array_print$(n4$[], "2 extra at front")

array.copy str$[2,3], n5$[]
n$=array_print$(n5$[], "trim to middle three"

array.copy str$[1,2],n6$[-1]
n$=array_print$ (n6$[], "trim to first 2 and add 1 at start")

end
```

f27_list.bas

```
!!
This program demonstrates
the use of the LIST commands
!!

! A function to print any
! list

fn.def lprint(list, msg$)
print msg$
```

```
list.size list, size
if size = 0
 print "Empty list"
 print " "
 fn.rtn 0
 endif

list.type list,type$
if type$ = "N"
 for i = 1 to size
  list.get list, i, num
  print num
 next i
else
 for i = 1 to size
  list.get list, i, str$
  print str$
 next i
endif
print " "
fn.rtn size
fn.end

list.create N, num
x = lprint(num, "Newly created empty list")

! Load an array and
! add the array to
! empty list

array.load n[], 1, 2, 3, 4
list.add.array num,n[]
x = lprint(num, "Array added to empty list")

! Add an element

list.add num, 5
x = lprint(num, "Element 5 added to list")

! Remove and element from the list

list.remove num, 2
x = lprint(num, "Element 2 removed from list");

! Insert a new element 2

list.insert num, 2, 22
x = lprint(num, "New element inserted at 2")

! Replace an element

list.replace num,2, 222
x = lprint(num, "Element 2 replaced")

! Add a list

list.add.list num, num
x = lprint(num, "List added to itself")
```

```
! List copied to new array

list.toarray num, array[]
array.length length, array[]
print "List copied to new array"
for i = 1 to length
 print array[i]
next i
print " "


! Clear the list

list.clear num
x = lprint(num, "List cleared")

! All of the operations
! can be done with strings
!

array.load s$[], "a", "b", "c"
list.create S, str
list.add.array str, s$[]
x = lprint (str, "A string list")

END
```

F28_bundle.bas !!

```
This program illustrates
the various Bundle commands
!!

! A function to print a
! any Bundle

fn.def bprint(bundle, msg$)
print msg$

! get the list of the keys
! in this bundle
! and the number of keys
! if the numbers of keys
! is zero then the bundle
! has no keys

bundle.keys bundle, list
list.size list, size
if size = 0
 print "Empty bundle"
 print " "
 fn.rtn 0
endif

! For each key,
! get the key type
```

```
! and then get key's
! value base upon
! the type

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

print " "
fn.rtn 1
fn.end

! Create a bundle and print it

bundle.create b1
x = bprint(b1, "A new, empty bundle")

! put some keys and values

bundle.put b1, "First_Name", "Frank"
bundle.put b1, "Last_Name", "Smith"
bundle.put b1, "Age", 44
bundle.put b1, "Employee_Number", 163
x = bprint(b1, "B1 with keys and values")

! Change the value of the Age key

bundle.put b1, "Age", 45
x = bprint(b1, "B1 with Age Value changed")

END
```

F28_Bundles.bas

```
!!
This program illustrates
the various Bundle commands
!!

! A function to print a
! any Bundle

fn.def bprint(bundle, msg$)
print msg$

! get the list of the keys
! in this bundle
! and the number of keys
! if the numbers of keys
! is zero then the bundle
```

```
! has no keys

bundle.keys bundle, list
list.size list, size
if size = 0
 print "Empty bundle"
 print " "
 fn.rtn 0
endif

! For each key,
! get the key type
! and then get key's
! value base upon
! the type

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

print " "
fn.rtn 1
fn.end

! Create a bundle and print it

bundle.create b1
x = bprint(b1, "A new, empty bundle")

! put some keys and values

bundle.put b1, "First_Name", "Frank"
bundle.put b1, "Last_Name", "Smith"
bundle.put b1, "Age", 44
bundle.put b1, "Employee_Number", 163
x = bprint(b1, "B1 with keys and values")

! Change the value of the Age key

bundle.put b1, "Age", 45
x = bprint(b1, "B1 with Age Value changed")

END
```

f29_stack.bas

```
!!
This program demonstrates
the operation of Stacks
```

```
!!

! Create a numeric stack
stack.create N, nStack

! Push some numbers onto
! the numeric stack

For i = 1 to 5
 stack.push nStack, i
 print "Push "; i
next i

! Pop numbers from
! stack until empty

stack.isempty nStack, empty
while !empty
 stack.pop nStack, value
 print "Pop, top value = "; value
 stack.isempty nStack, empty
repeat

print "Stack is empty"
print " "
! Create a string stack
stack.create S,sStack

! Push some strings onto
! the string stack

s$ = "Last"
gosub sPush
s$ = "In"
gosub sPush
s$ = "First"
gosub sPush
s$ = "Out"
gosub sPush

stack.isempty sStack, empty
while !empty
 stack.pop sStack, value$
 print "Pop, top value = "; value$
 stack.isempty sStack, empty
repeat

print "Stack is empty"
END

sPush:
print "Push "; s$
stack.push sStack, s$
return
```

f30_poly.bas

```
!!
```

```
This program demonstrates
the gr.poly command.

Several polygons with
different numbers of
sizes will be drawn.
!!

! Set up the graphics
gr.open 255, 255, 255, 255
gr.set.stroke 4
gr.orientation 0  % Force landscape

! Get the screen size
! of this device and then
! scale from 800 x 480
! to fit this screen

gr.screen w, h
dw = 800
dh = 480
sw = w/dw
sh = h/dh
gr.scale sw,sh

! Draw a filled triangle
! and then move it and
! draw it is unfilled.

array.load a[], 60, 20~
180, 70~
30, 180

list.create n, List1
list.add.array List1, a[]

gr.color 255, 255, 0, 0, 1
gr.poly pt, List1
gr.color 255, 255, 0, 0, 0
gr.poly pt, List1, 200, 0

! Draw four sided, regular
! filled polygon and then
! move it and draw it as
! unfilled

array.load b[], 500,20~
700, 20~
550, 100~
350, 100

list.create n, List2
list.add.array List2, b[]
gr.color 255, 255, 0, 0, 1
gr.poly pt, List2
gr.color 255, 255, 0, 0, 0
gr.polyt pt, List2, 0, 110
```

```
! Draw a polygon whose
! lines cross

array.load c[], 50, 300~
250, 300~
80, 400~
220, 400

list.create n, List3
list.add.array List3, c[]
gr.color 255, 255, 0, 255, 1
gr.poly pt, List3

! Draw an irregular,
! six sided, unfilled
! polygon

array.load d[], 550,260~
650, 260~
750, 320~
650, 420~
550, 420~
450, 320

list.create n, List4
list.add.array List4, d[]
gr.color 255, 0, 0, 255, 0
gr.poly pt, List4

gr.render


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

f31_socket_time

```
! Demo TCP Client by getting
! current time from Time Server
! in Mt. View, CA

! Other time servers can be found at
! http://tf.nist.gov/tf-cgi/servers.cgi

socket.client.connect "207.200.81.113", 13

! Wait for first line (a blank line)
! or time out after 10 seconds

maxclock = clock() + 10000
do
 socket.client.read.ready flag
 if clock() > maxclock
   print "Read time out"
```

```
   end
 endif
until flag

! Read the blank line and print it

socket.client.read.line line$
print line$

! Wait for the time data line
! Read it. Print it.

socket.client.read.line line$
print line$

! Protocol done. Close Client

socket.client.close

end
```

f32_tcp_ip_socket.bas

```
!!

******  READ!  *******

Sample program demonstrating
TCP/IP sockets. The program
has two parts: a server side
and a client side. Both sides
need to be running on different
Android devices.

The Server will wait for a Client
to connect to it. When the
connection is made, the Server
will wait for a message from
the Client. Once the message
has been received, the Server
will send a message back
to the Client.

Some devices/networks do not
allow smart phone servers onto
their data network.

In that case, the server must be
inside a LAN.

LAN = Local Area Network (WiFi)
WAN = Wide Area Network (Internet)

If the Client is on the same LAN
as the Server, the Client can
connect directly to the Server
using the Server's LAN IP.
```

If the Client is outside of the
Server LAN then the Client must
must connect using the Server's
WAN IP.

It will be necessary to program
the Server's LAN router to pass
a specific port through from the
WAN to Server's LAN IP

!!

```
array.load type$[], "Server", "Client"
msg$ = "Select TCP/IP socket type"
select type, type$[], msg$
if type = 0
 "Thanks for playing"
 end
elseif type = 2
 goto doClient:
endif

!***********  Server Demo  **************

input "Enter the port number", port, 1080

socket.myip ip$
print "LAN IP: " + ip$
graburl ip$, "http://automation.whatismyip.com/n09230945.asp"
print "WAN IP: " + ip$

! Create the server on the specified port
socket.server.create port

newConnection:

! Connect to the next Client
! and print the Client IP
Print "Waiting for client connect"
socket.server.connect
socket.server.client.ip ip$
print "Connected to ";ip$

! Connected to a Client
! Wait for Client to send a message
! or time out after 10 seconds

maxclock = clock() + 10000
do
 socket.server.read.ready flag
 if clock() > maxclock
   print "Read time out"
   end
 endif
until flag

! Message received. Read it.
! Print it
```

```
socket.server.read.line line$
print line$

! Send a message back to client

socket.server.write.line "Server to client message"

! Finished this protocol
! Disconnect from Client

socket.server.disconnect
print "Disconnected from client"

! Loop to get the next Client

goto newConnection

! ****************** Client Demo *****************

doClient:

input "Enter the connect-to IP", ip$
input "Enter the port number", port, 1080

! Connect to the specified IP on the
! specified Port

clientAgain:

socket.client.connect ip$, port
print "Connected"

! When the connection is established,
! send the server a message

socket.client.write.line "Client to server message"

! and then wait for Server to respond
! or time out after 10 seconds

maxclock = clock() + 10000
do
 socket.client.read.ready flag
 if clock() > maxclock
   print "Read time out"
   end
 endif
until flag

! Server has sent message.
! Read it. Print it.

socket.client.read.line line$
print line$

! Close the client
```

```
socket.client.close
print "Disconnected from server"

input "Connect again? Y or N", again$, "Y"
if again$ = "Y" then goto clientAgain

end
```

f33_camera.bas

```
!!
This program demonstrates the
use of the camera interfaces
by BASIC!

Note: The captured images are
not saved.
!!

! Load an array with the
! Select Options
array.load method$[], "Quit" ~
"Use Device User Interface" ~
"Automatic with auto flash", "Automatic mode without flash", "Automatic mode with flash"~
"Manual with auto flash", "Manual mode without flash", "Manual mode with flash"

! Set the Popup Message
msg$ ="Select A Capture Mode"

! Open Graphic
gr.open 255, 0, 0, 0 % Black Background

again:

!Reset orientation to Landscape

gr.orientation 0 % Force Landscape

! Ask the user what to do
select mode, method$[], msg$

! A choice has been made
! Act on that choice
sw.begin mode

sw.case 0 % Back Key
sw.case 1 % Quit
Print "Done taking pictures"
end
sw.break

sw.case 2 % Use Device User Interface
gr.camera.shoot bm_ptr
sw.break
```

```
sw.case 3 % Automatic mode auto flash
gr.camera.autoshoot bm_ptr, 0
sw.break

sw.case 4 % Automatic mode without flash
gr.camera.autoshoot bm_ptr, 2
sw.break

sw.case 5 % Automatic mode with flash
gr.camera.autoshoot bm_ptr, 1
sw.break

sw.case 6 % Manual mode with auto flash
gr.camera.manualshoot bm_ptr, 0
sw.break

sw.case 7 % Manual mode without flash
gr.camera.manualshoot bm_ptr, 2
sw.break

sw.case 8 % Manual mode with flash
gr.camera.manualshoot bm_ptr, 1
sw.break

sw.end

! Test to insure that an
! image was captured
if bm_ptr = 0
 print "Image not captured"
 goto again:
endif

! If the picture is in
! in Portrait mode, switch
! to Portrait mode

! Scale the bitmap to fit the
! screen with the proper
! aspect ratio

gr.bitmap.size bm_ptr, bw,bh
if bh > bw
 gr.orientation 1
 gr.screen sw, sh
 ar = bw/bh
 gr.bitmap.scale scaled_bm, bm_ptr, sw, sh*ar
else
 gr.screen sw, sh
 ar = bh/bw
 gr.bitmap.scale scaled_bm, bm_ptr, sw*ar, sh
endif

! Draw the scaled image bitmap
gr.bitmap.draw obj_ptr, scaled_bm, 0, 0

! Add some text
```

```
gr.text.size sw/25
gr.text.align 1
xleft = sw/12
xtop = sh - 2*(sw/25)
gr.color 255, 255, 255, 255, 1
gr.text.draw P, xleft,  xtop, "Tap screen to take another picture."

! Now render the picture and text
gr.render

! Wait until touch and then untouch

flag = 0
do
 gr.touch flag, xx, yy
 until flag
do
 gr.touch flag, xx, yy
 until !flag

!Clear the screen

gr.cls
gr.render

! Delete the old bitmaps
gr.bitmap.delete bm_ptr
gr.bitmap.delete scaled_bm

! Ready for the next capture
goto again
```

f34_remote_camera.bas

```
!!

******  READ!  *******

See f32_tcp-ip_socket.bas
introduction for details
about setting up a server
and a client.

This program must be run
on two different Android
devices.

In this program, it is
a server that takes a picture
under command from a client. It
is a client that retrieves the
image and displays it.

The viewer saves the received
image in
/sdcard/rfo-basic/data/rimage.jpg
and displays the image
```

```
!!

array.load type$[], "Remote Camera", "Remote Viewer"
msg$ = "Select Remote Type"
select type, type$[], msg$
if type = 0
 "Thanks for playing"
 end
elseif type = 2
 goto doViewer:
endif

!***********  Camera Server Server Demo  **************

ServerErrorCount = 0
ServerMaxError = 100
RecoveryState = -1;

input "Enter the port number", port, 1080

socket.myip ip$
print "LAN IP: " + ip$
graburl ip$, "http://automation.whatismyip.com/n09230945.asp"
print "WAN IP: " + ip$

! Create the server on the specified port
socket.server.create port
RecoveryState = 0

newConnection:

! Connect to the next Client
! and print the Client IP
print ""
Print "Waiting for viewer client connect"
socket.server.connect
socket.server.client.ip ip$
print "Connected to ";ip$
RecoverState = 0

! Connected to a Client
! Wait for Client to send a message
! or time out after 10 seconds

maxclock = clock() + 60000
do
 socket.server.read.ready flag
 if clock() > maxclock
   print "Read time out"
   goto OnError
 endif
until flag

! Message received. Read it.

socket.server.read.line msg$
print "Viewer command = ";msg$
```

```
! The message is a string with
! single digit

! Open Graphics
gr.open 255, 0, 0, 0 % Black Background

sw.begin msg$

sw.case "0" % 0 = Quit
 Print "Viewer says to shutdown"
 end
 sw.break


sw.case "1" % Automatic mode auto flash
 gr.camera.autoshoot bm_ptr, 0
sw.break

sw.case "2" % Automatic mode without flash
 gr.camera.autoshoot bm_ptr, 2
sw.break

sw.case "3" % Automatic mode with flash
 gr.camera.autoshoot bm_ptr, 1
sw.break

sw.default
 print "Do some more debugging"
 end

sw.end

! We do not need the bitmap,
! so delete it.

gr.bitmap.delete bm_ptr

! Close graphics
gr.close

! The autoshoot mode places an
! image file, image.jpg in the
! /sdcard/rfo-basic/data/
! directory
!
! We will send that image to
! the client

byte.open R, fr, "image.jpg"
print "Sending image"
socket.server.write.file fr
print "Image sent"

! Finished this capture.
! Disconnect from Client

socket.server.disconnect
```

```
print "Disconnected from viewer"

! Loop to get the next Client

goto newConnection

! Server Error Recovery Code

onError:

! If not in server mode, quit
if type <> 1
 print "Client error. Terminating"
 end
endif

if RecoverState = -1
 Print "Device not connected to LAN or WAN"
 END
endif

! Limit the number of server errors
! to avoid endless looping


if RecoverState = 0
        ServerErrorCount = ServerErrorCount + 1
        Print "Server Error. Recovering " + str$(ServerErrorCount)
        if ServerErrorCount > ServerMaxError
         print "Exceeded Max Server Error Count. Terminating"
        end
endif

! Try to disconnect and close
! Server. If one fails, do
! not try it again until
! and Server is created

if RecoverState = 0
 Print "Attempting to disconnect"
 RecoverState = 1
 socket.server.disconnect
 print "Disconnected"
 goto newConnection
elseif
 Print "Attempting to close Server"
 RecoverState =2
 socket.server.close
 Print "Closed"
endif

! Try to crate and new server.

Print "Attempting to create new Server"
socket.server.create port
RecoveryState = 0
goto newConnection
```

```
! ****************** Client Demo ****************

doViewer:

! Load an array with the
! Select Options
array.load method$[], "Quit" ~
"Capture with auto flash", "Capture without flash", "Capture with flash"~
"Shut down remote camera"

! Set the Popup Message
msg$ ="Connected. Select A Capture Mode"

input "Enter the connect-to IP", ip$
input "Enter the port number", port, 1080


viewerAgain:

! Connect to the specified IP on the
! specified Port

socket.client.connect ip$, port
print ""
print "Connected to Camera Server"


again:

! Ask the user what to do
select mode, method$[], msg$

! A choice has been made
! Act on that choice
sw.begin mode

sw.case 0 % Back Key
sw.case 1 % Quit
 Print "Done taking pictures"
 end
sw.break

sw.case 2 % Capture with auto flash
 socket.client.write.line "1"
sw.break

sw.case 3 % Automatic mode without flash
 socket.client.write.line "2"
sw.break

sw.case 4 % Automatic mode with flash
 socket.client.write.line "3"
sw.break

sw.case 5 % Shut down camera
 socket.client.write.line "0"
 print "Done taking pictures"
 socket.client.close
```

```
end
sw.break

sw.end

! and then wait for Server to respond
! or time out after 30 seconds

Print "Waiting for image"
maxclock = clock() + 30000
do
 socket.client.read.ready flag
 if clock() > maxclock
   print "Read time out"
   end
 endif
until flag

! Server has sent the image file.
! Read it. Print it.

byte.open W, fw, "rimage.jpg"
socket.client.read.file fw
byte.close fw
Print "Image received"

! The image is now in the file
! "rimage.jpg"

! Close the client

socket.client.close
print "Disconnected from server"

! Open Graphics
gr.open 255, 0, 0, 0 % Black Background

!Reset orientation to Landscape
gr.orientation 0 % Force Landscape

! Load the file into a bitmap

gr.bitmap.load bm_ptr, "rimage.jpg"

! Scale the bitmap to fit the
! screen with the proper
! aspect ratio

gr.bitmap.size bm_ptr, bw,bh
if bh > bw
 gr.orientation 1
 gr.screen sw, sh
 ar = bw/bh
 gr.bitmap.scale scaled_bm, bm_ptr, sw, sh*ar
else
 gr.screen sw, sh
 ar = bh/bw
 gr.bitmap.scale scaled_bm, bm_ptr, sw*ar, sh
```

```
endif

! Draw the scaled image bitmap
gr.bitmap.draw obj_ptr, scaled_bm, 0, 0

! Add some text

gr.text.size sw/25
gr.text.align 1
xleft = sw/12
xtop = sh - 2*(sw/25)
gr.color 255, 255, 255, 255, 1
gr.text.draw P, xleft,  xtop, "Tap screen to take another picture."

! Now render the picture and text
gr.render

! Wait until touch and then untouch

flag = 0
do
 gr.touch flag, xx, yy
 until flag
do
 gr.touch flag, xx, yy
 until !flag

! Delete the old bitmaps
gr.bitmap.delete bm_ptr
gr.bitmap.delete scaled_bm

! Close graphics
gr.close

! Ready for the next capture
goto viewerAgain
```

f35_bluetooth.bas

```
!!
This program demonstrates using
Bluetooth as both an application
that listens for a connection
or makes a connection to a listener.

Before running this program use
the Android "Settings" application
to enable Bluetooth and to pair
with the device(s) that you will
talk to.

This program will send and receive
data bytes to and from a connected
device.

You can use this program to
(among many other things) set
```

up a chat between two Android
devices.

Note: The UUID used by default is
the standard serial port UUID. This
can be changed. Read the manual.

!!

! Begin by opening Bluetooth
! If Bluetooth is not enabled
! the program will stop here.

bt.open

! When BT is opened, the program
! will start listening for another
! device to connect. At this time
! the user can continue to wait
! for a connection are can attempt
! to connect to another device
! that is waiting for a connection

! Ask user what to do

array.load type$[], "Connect to listener", "Continue to listen for connection", "Quit"
msg$ = "Select operation mode"

new_connection:

select type, type$[], msg$

! If the user pressed the back
! key or selected quit then quit
! otherwise try to connect to
! a listener

```
if (type = 0) | (type =3)
 print "Thanks for playing"
 bt.close
 end
elseif type = 1
 bt.connect
endif
```

! Read status until
! a connection is made

```
Do
 bt.status s
 if s = 1
  print "Listening
  elseif s =2
   print "Connecting"
   elseif s = 3
    print "Connected"
 endif
```

```
 pause 1000

until s =3

! When a connection is made
! get the name of the connected
! device

bt.device.name device$

! *** Read/Write Loop ****

RW_Loop:

Do

! Read status to insure
! that we remain connected.
! If disconnected, program
! reverts to listen mode.
! In that case, ask user
! what to do.

 bt.status s
 if s<> 3
  print "Connection lost"
  goto new_connection
 endif

! Read messages until
! the message queue is
! empty

 Do
  bt.read.ready rr
  if rr
   bt.read.bytes msg$
   print device$;": ";msg$
  endif
 until rr = 0

! Message Queue is empty
! Send a message
! Note: If no text is entered
! then no text will be sent

 Input "Text to send", msg$
 if (msg$ <> "") & (msg$<> "*")
  bt.write msg$
  print "Me: "; msg$
 endif

until msg$ = "*"

! if msg$ is "*" then close connection
! and restart

bt.close
```

```
bt.open
goto new_connection
```

f36_superuser.bas

```
!!
The program implements a
Superuser terminal.

Your device must be rooted
to successfully run this
program.

!!

! Get Superuser permission
su.open

! Main loop
loop:

! get the command from the user
tget r$, "cmd: "

! if cmd is "exit"
if r$ = "exit"
 su.close
 print "Exited"
 end
endif

! write the command
su.write r$

! Give system time to respond
pause 500

! check for a response
su.read.ready ready

! if no input, do next command
if !ready then goto loop

! read responses
do
 su.read.line l$
 print l$
 su.read.ready ready
until !ready

! Go for the next command
goto loop
```

f37_html_demo.bas

```
!!
This program demonstrates some
```

```
of the possibilities
when using the BASIC! html commands.
!!

! html must be opened before doing
! any html command
html.open

! Load the file, htmlDemo1.html

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

!The user now sees the html

! We can now monitor the user
! actions

xnextUserAction:

! loop until data$ is not ""

do
html.get.datalink data$
until data$ <> ""

! The first four characters of data$
! identify the type of data returned.
! Extract those three characters to
! use in a switch statement

type$ = left$(data$, 4)

! Trim the first four characters from
! data$

data$ = mid$(data$,5)

! Act on the data type
! Shown are all the current data types

sw.begin type$

! Back Key hit.
! if we can go back then do it
 sw.case "BAK:"
  print "BACK key: " + data$
  if data$ = "1" then html.go.back else end
  sw.break

 ! A hyperlink was clicked on
 sw.case "LNK:"
   print "Hyperlink selected: "+ data$
   html.load.url data$
   sw.break

 ! An error occured
 sw.case "ERR:"
   print "Error: " + data$
```

```
   sw.break

  ! User data returned
  sw.case "DAT:"
    print "User data: " + data$
                if  left$(data$, 4) = "Exit"
                 print "User ended demo."
     html.close
                 end
                 endif
               sw.break

! Form data returned.
! Note: Form data returning
! always exits the html.

sw.case "FOR:"
 print "Form data: "+data$
 END
 sw.break

! download requested
! extract the filename
! tell user download starting
! download it

sw.case "DNL:"
  print "Download: " + data$
  array.delete p$[]
  split p$[], data$, "/"
  array.length l, p$[]
  fn$ = p$[l]
  html.load.string "<html> Starting download of " + fn$ + "</html>"
  byte.open r,f,data$
  pause 2000
  html.go.back
  byte.copy f,fn$
  byte.close
  sw.break


sw.default
 print "Unexpected data type:", data$
 END

 sw.end

 goto xnextUserAction
```

htmldemo1.html

```
<html><body>

<!--
```

htmlDemo1.html

```
This simple Javascript Function sends
the string "data" to the the excuting BASIC!
program.

The BASIC command, "html.getdatalink data$", is
used to read linked data. If data$ is empty ("") then
not data has been sent.  The BASIC! programmer
can loop on this value until a non-empty string is
returned.

This function is the heart the HTML/BASIC! interactive
interface.
-->

<script type="text/javascript">
   function doDataLink(data) {
      Android.dataLink(data);
   }
</script>

<!--
Show some marked up text
-->

<b>HTML Demo #1</b>
<p>

<!--
Display an image located in:
/sdcard/rfo-basic/data/
-->
<img src="Cartman.png">
<p>

<!--
Display a hyperlink to the next Demo.
The link file is located in
/sdcard/rfo-basic/data/
-->

<a href="htmlDemo2.html">Go to Demo #2 </a>

<p>
```

```
<!--
Display a button labeled Exit Demo. Pressing this button
<html><body>

<!--

htmlDemo1.html

This simple Javascript Function sends
the string "data" to the the excuting BASIC!
program.

The BASIC command, "html.getdatalink data$", is
used to read linked data. If data$ is empty ("") then
not data has been sent.  The BASIC! programmer
can loop on this value until a non-empty string is
returned.

This function is the heart the HTML/BASIC! interactive
interface.
-->

<script type="text/javascript">
   function doDataLink(data) {
      Android.dataLink(data);
   }
</script>

<!--
Show some marked up text
-->

<b>HTML Demo #1</b>
<p>

<!--
Display an image located in:
/sdcard/rfo-basic/data/
-->
<img src="Cartman.png">
<p>

<!--
Display a hyperlink to the next Demo.
The link file is located in
```

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

<a href="htmlDemo2.html">Go to Demo #2 </a>

<p>
Do a download.
<p>
<a
href="http://laughton.com/basic/versions/v01.54/Basic.apk">http://laughton.com/basic/versions/v
01.54/Basic.apk</a>

<p>

<!--
Display a button labeled Exit Demo. Pressing this button
send a dataLink message to BASIC!

When the BASIC! program gets this message,
it will close  HTML .
-->

        <input type="button" value="Exit Demo" onClick="doDataLink('Exit Demo')" />


<P>

<!--
Hitting the BACK key at this point  will exit html because
this is start of thistory chain
-->

You can also exit the Demo by hitting the BACK key


</body></html>




-->

You can also exit the Demo by hitting the BACK key

</body></html>
```

htmlDemo2.html

```
<html>

<head >

 <meta http-equiv="content-type" content="text/html;charset=UTF-8"/>

 <title >Demo # 2, Forms</title>

</head>

<!--
This simple Javascript Function sends
the string "data" to the the excuting BASIC!
program.

The BASIC command, "html.getdatalink data$", is
used to read linked data. If data$ is empty ("") then
not data has been sent.  The BASIC! programmer
can loop on this value until a non-empty string is
returned.

This function is the heart the HTML/BASIC! interactive
interface.
-->

<script type="text/javascript">
   function doDataLink(data) {
      Android.dataLink(data);
   }
</script>

<body bgcolor="White">

 <div align="center">
 <!--
Display an image located on the internet
-->

<img src="http://laughton.com/basic/logo.png">
<p>

   <h1><b>Fake Sign In</h1>

   <form id='main' method='get' action='FORM'>
```

```
    <p>Name: <input type='text' name='id'/></p>

    <p>Email: <input type='text' name='password'/></p>

    <p><input type='submit' name='submit' value='sign_in'/></p>

  </form>

<p>
<!--
Display a button labeled Exit Demo. Pressing this button
send a dataLink message to BASIC!

When the BASIC! program gets this message,
it will close this HTML.
-->

        <input type="button" value="Exit Demo" onClick="doDataLink('Exit Demo')" />

    <p>

<!--
Hitting the BACK key at this point will take you back to the privious
entry in the history tree. In this case that is htmlDemo1.html
-->
You can go back to Demo #1 by hitting the BACK key.


 </div>

</body>

</html>
```

F38_html_edit.bas

```
!! htmledit.bas
A program to edit html files.
Input the filename without
the .html extension
!!

data$ = ""

! Get the filename
```

```
! and add the .html

input "Enter filename", fn$
fn$=fn$+".html"

! test to see if the file exists
! if it does exist then
! open it, grab the contents
! and close the file

file.exists fe, fn$
if fe then grabfile data$, fn$

! use text.input to do
! the editing

text.input data1$, data$

! Now write the edited
! text to the file

 text.open w,f,fn$
 text.writeln f,data1$
 text.close f

 end
```

f39_downloader.bas

```
!* Download Helper Plus! ver. 2.2 *
! f39_downloader.bas
!
! Downloads shared BASIC! programs
! from the BASIC! shared program ftp site.
! Run the program and select the BASIC!
! program that you want to down load.
!
! Thank you, Droid_Mike, for this program

URL$ = "http://laughton.com"
let source_path$ =  URL$ + "/basic/programs/"

cls
print "Welcome to the RFO Basic! Download Helper Plus! ver 2.1"
print "Please wait while the file list is being downloaded."
print ""

relist:

GrabURL result$, source_path$
split pre_list$[], result$, "<a href=" + chr$(34)
```

```
List.create S, program_list

array.length ln, pre_list$[]
for i= 1 to ln
      List.add program_list, left$(pre_list$[i], is_in(chr$(34),
pre_list$[i])-1)
next i

! Have a choice to quit at end
List.add program_list, "QUIT!"

! OK, now list all the filenames
List.size program_list, slist

! First 7 lines are junk, so we will skip them
line_pad = 7

List.ToArray program_list, pre_show_array$[]
Array.copy pre_show_array$[line_pad], show_array$[]

Array.length length, show_array$[]
for i = 1 to length
      ! Make sure spaces are displayed accurately and dir's are skipped
      temp_name$ = replace$(show_array$[i],"%20"," ")
      show_array$[i]=temp_name$
next i

do
      select dfnum, show_array$[], "Choose File to Download"
      List.get program_list, line_pad-1+dfnum, dfname$

      ! see if user quits
      if dfname$ = "QUIT!" then end

      ! see if it is a directory
      if is_in("/",dfname$) > 0
            ! See if user clicked on previous directory
            if is_in("/basic/programs/",dfname$) then
                  let source_path$ = URL$ + dfname$
            elseif is_in("/basic/",dfname$) then
                  let source_path$ = URL$ + "/basic/programs/"
                  else
                  let source_path$ = source_path$ + dfname$
            endif
            undim pre_show_array$[]
            undim show_array$[]
            undim pre_list$[]
            List.clear program_list
            print "Now loading directory "; source_path$
            goto relist
      endif

      ! guess the destination directory based on the filename
      default_dest$ = "../data/"
      if is_in(".bas", dfname$) > 0 then default_dest$ = "../source/"
      if is_in(".db", dfname$) > 0 then default_dest$ = "../databases/"
```

```
      input "Enter Destination Directory", ddir$, default_dest$

      print ""
      print "Downloading "; source_path$; " to "; ddir$ + dfname$

      byte.open R, FN, source_path$+dfname$
      byte.copy FN, ddir$+dfname$

      print "FILE DOWNLOAD COMPLETE!"

      pause 2000

until 0

end
```

Appendix C – Launcher Short Cuts 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.

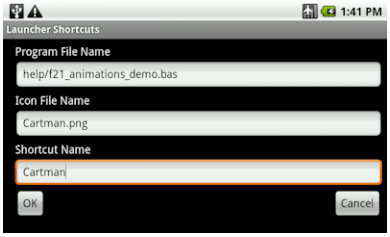
There is also an option to actually build a standalone application .apk file that does not require the BASIC! application to be installed. The process is more difficult but it will result in an application that can be offered on the Android Market. See Appendix D.

The procedure for making a Shortcut Application

1. Start BASIC!
2. Press 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. Press 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

What you need to know

1. The icon image file must be located in the "/sdcard/rfo-basic/data/" directory.

2. 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.

3. 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.

4. 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 .

5. The Shortcut name should be nine (9) characters or less. Android will not show more than nine characters.

6. You can create as many shortcuts as you home screen(s) can handle.

7. Pressing "Cancel" in the Launcher Shortcuts dialog will simply cancel the operation and return to the

home screen.

8. If you plan to use a BASIC! Launcher Shortcut, you should always exit BASIC! using Menu->More->Exit. If a Launched program is running, pressing BACK once or twice will exit BASIC back to the Home Screen.

Appendix D – Building a Standalone Application

Note: A very clever BASIC! user has automated this entire process. You can now accomplish all this in just about one click. For details, see

http://rfobasic.freeforums.org/rfo-basic-app-builder-t695.html

Thank you, Mr. **mougino.**

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.

Setting Up the Development Environment

The complete Android development environment requires the downloading and installation of Java, Eclipse and the Android SDKs. A complete set of instructions for doing this can be found on the Android Developer's Web site at:
http://developer.android.com/sdk/installing.html.

After the development environment has been installed, insure that Android Preferences are setup in the Eclipse. Select Window -> Preferences -> Android. Make sure that the Android SDK is properly pointed to in the SDK Location Field.

Download the Stand Alone Application Source Code

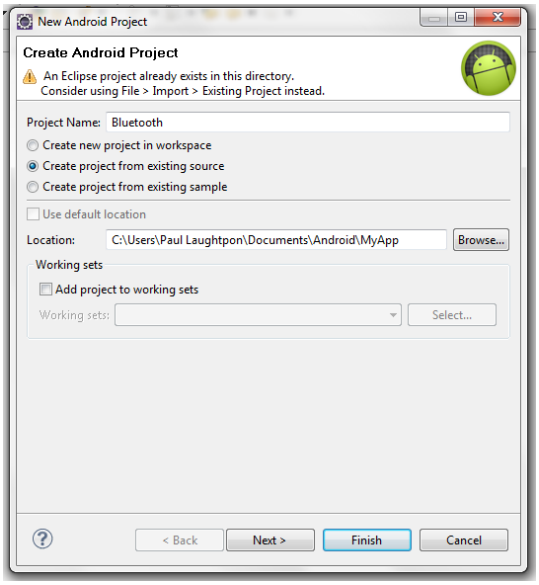
Each release starting with v01.09 will have a link to the Stand Alone Application source code for that particular release. The download file is named MyApp.zip. Unzip this file. There will be a readme.txt file and a folder, MyApp. MyApp contains the source code in the form of an Eclipse project. Move the folder to the location where it will want it to reside long term. The contents of the files in this folder will be changed as you make changes to the code in Eclipse.

The readme.txt file will contain any information uniquely pertaining to that particular release. In particular it will list the source files that have changed since the previous release. This information will

be useful when upgrading the application to use a new release of BASIC!. You will be making a number of changes to the original MyApp source code. Most new releases of BASIC! have only one source file that has changed. There is no need to repeat all of the original MyApp changes if only one file has changed in a new release. You can just copy that one file from the new release source and paste it into your MyApp. One implication of this process is that you will probably want to keep the original MyApp folder in a different directory than you will be using for future MyApp folders.

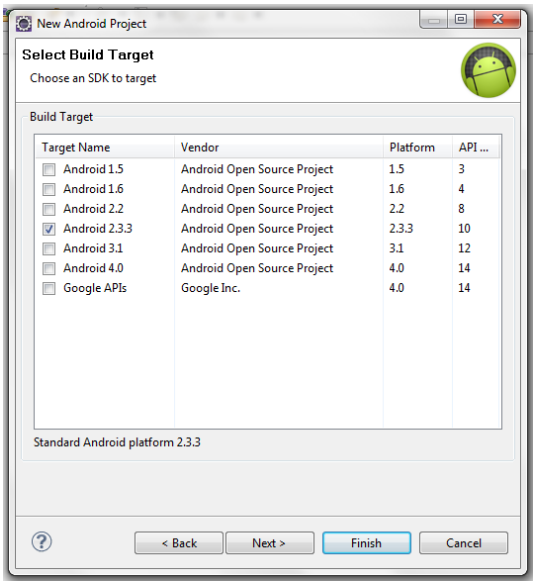
Create a New Project in Eclipse

Select File -> New -> Android Project



Give the project a name
Select the Create Project From Existing Source Option
Browse to the location of the MyApp Folder.

Press Next.



As of release 1.42 BASIC! will be built using Android 4.0.

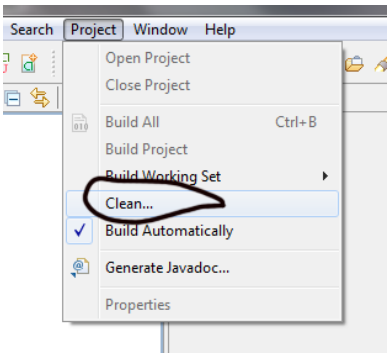
Select Android 4.0 from the build target rather than Android 2.3.3

Select Android 2.3.3 as the Build Target (No matter what firmware level your device is).
If you do not have Android 2.3.3 in this list, go to Window->Android SDK and AVD Manger->Available Packages and download 2.3.3

Press Finish.

Cleaning the Project

Select Project->Clean



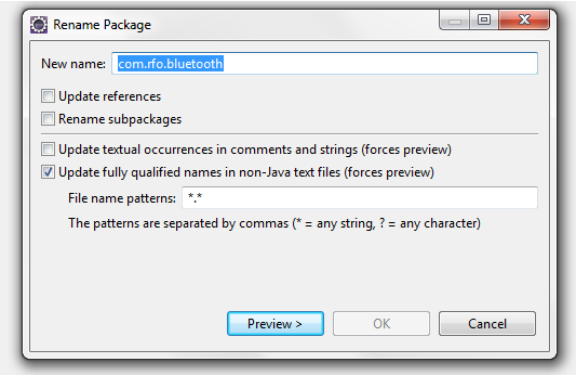
Press OK.

Now DO IT AGAIN. Clean the project a second time.

Rename the package

In the Package Explorer at the left side of the page, double click to Open "MyApp" and then "src".
Select com.rfo.myapp so that it is highlighted.
Select File -> Rename

Enter the new name for the package. Only change the part of the name after com.rfo.
Check the one box as shown.
Press Preview

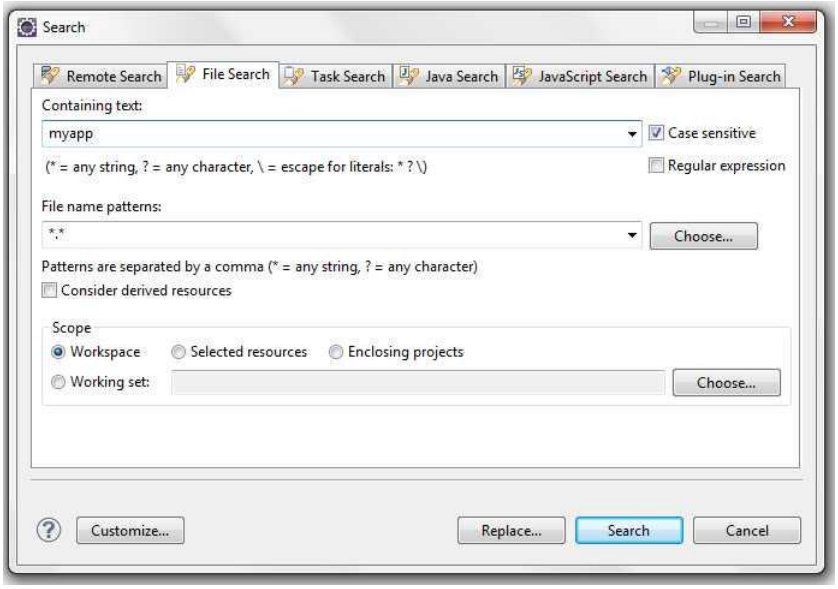


The Rename Preview dialog box will be shown. Press the OK button.
A Launch Configuration Dialog Box will be shown. Press Yes.

Next, select Search -> File.

Fill out the dialog box as shown:

- Enter myapp (lower case) into the Containing Text Field
- Check the Case Sensitive Box
- Enter *.* into the File Name Pattern



Press Replace.
The Replace Text Matches Dialog Box will be shown.
Enter the new name in the “With:” field.
Press OK.

Finally, Select Project -> Clean
Press OK on the Clean Dialog Box.

At this point the package has been successfully renamed. Next we will make the changes needed to run your BASIC! program

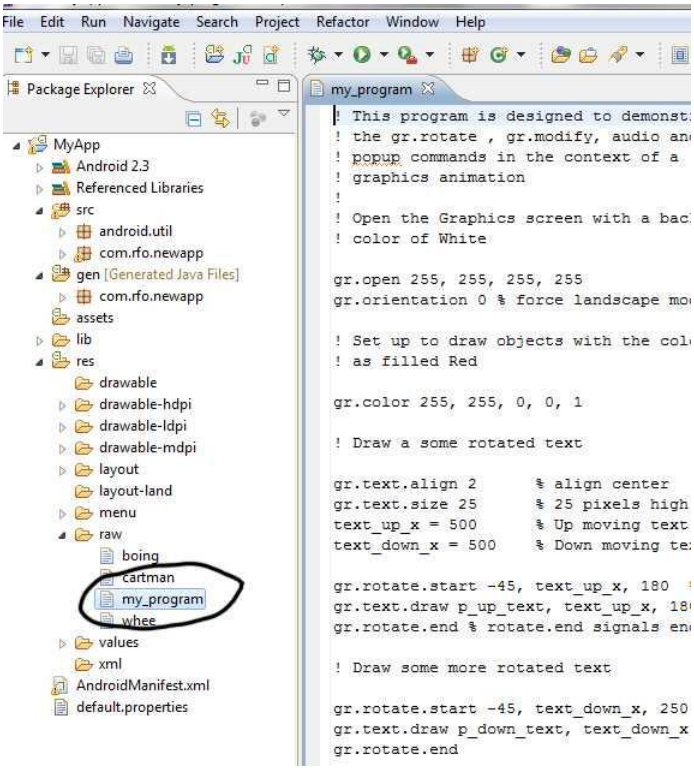
Installing A BASIC! Program Into the Application

Outside of Eclipse

Open you BASIC! program in a text editor
If your program uses INCLUDEd files, you should text-merge the included files into a single text file.
Copy the entire program into the Clipboard

In the Package Explorer on the left side,

Open res
Open raw
Double click on my_program
The file will be opened



Press Edit - > Select All
Press Edit -> Paste
Press the "X" on the my_program tab to close the file.
Press "Yes" on the Save Resource dialog box.

Making The Changes To Basic.java

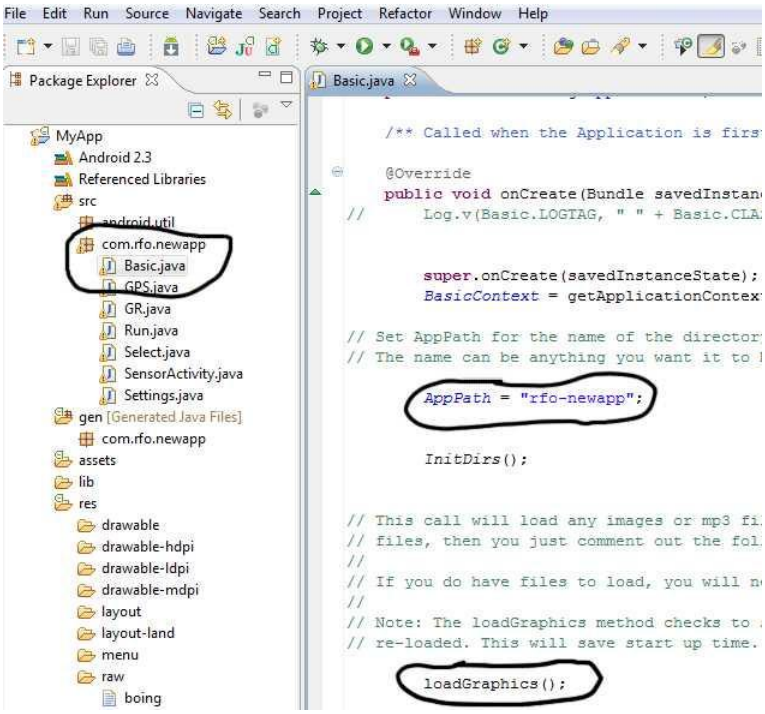
Basic.java is the Java Class that starts the execution of BASIC!. It receives control from the Android Launcher. The function of Basic.java is to initialize things. These things include setting up the application directories on the sdcard, loading your Basic program into execution memory and loading any image or audio files that may be needed. Once all this is done, control is transferred to Run.java to do the actual running of the program.

Open the Basic.java file

In the Package Explorer on the left side, open com.rfo.newapp.
Double click on Basic.java to open the file.

Naming the application top level file directory

If your application does any file I/O it will need an application specific directory in the root directory of the SDCARD.



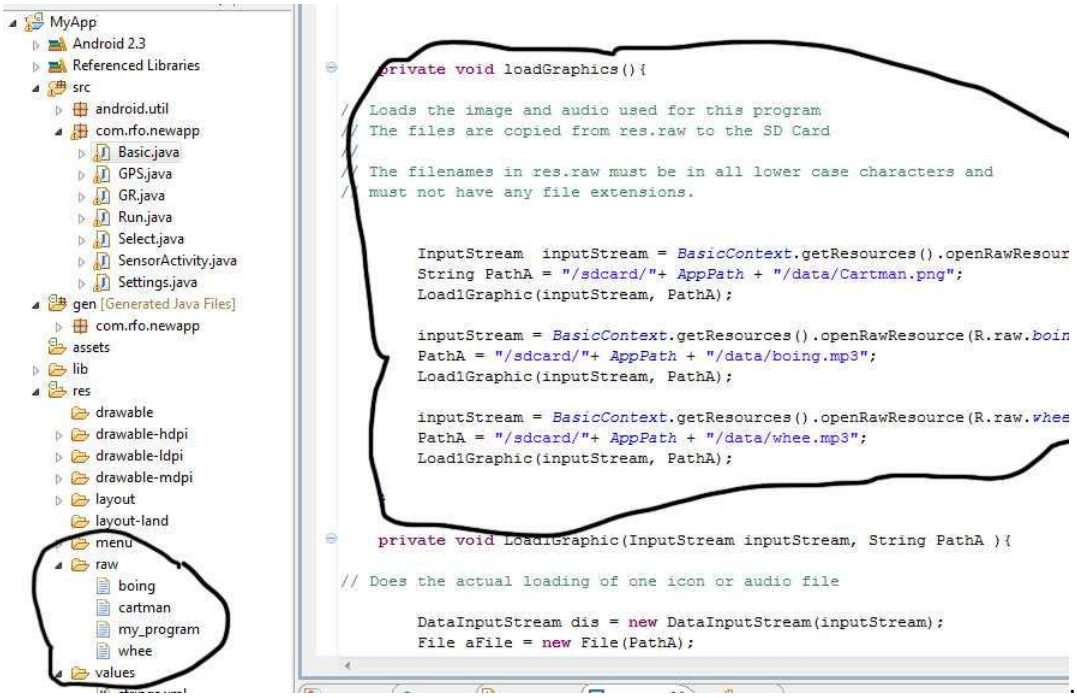
The name of this top level directory is set by the value in the AppPath variable. In this example, the name, "rfo-newapp" was set during the renaming process done above. You can use any name you wish for this directory.

Two sub-directories will be created under the directory. The "data" sub-directory will be used for your most of your files. The "databases" directory will be used for any SQLITE databases. These directories are created and initialized by the InitDirs() method call seen above.

Creating Image and Audio Files

Image, audio and other files that the program may need are delivered in the .apk file in res.raw. The data in these files is copied from res.raw and written into files in the applications "data" sub-directory. This copy work is done by the loadGraphics() method.

If your program does not need to create any files then you can simply comment out the loadGraphics() method call.



If you have files that you want to load, proceed as follows:

Outside of Eclipse, select and copy the file.
In Eclipse, right click on res.raw and select Paste
The file will be pasted into res.raw with the copied file name and extension.
You must rename the pasted file such that it has all lower case characters and does not have any file extension.
To do this, select the pasted file and then Select File -> Rename...

Once the files have been loaded into res.raw and renamed, you will change the code in loadGraphics.

For each file, you will need to change the file res.raw source name in the InputStream code line and the
destination file name in the PathA code line.
You may have cut and paste similar lines of code if you have more than three files to load.

Note: If you decide to delete the res.raw files, cartman, boing and whee, you will need to comment out the references to them in the loadGraphics code.

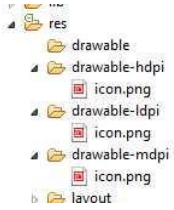
At this point you are done with the modifications to Basic.java. Click the X in the tab to save and close the file.

Be sure to check the Problems tab at the bottom of Eclipse to make you sure you have not made any coding mistakes in Basic.java.

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:

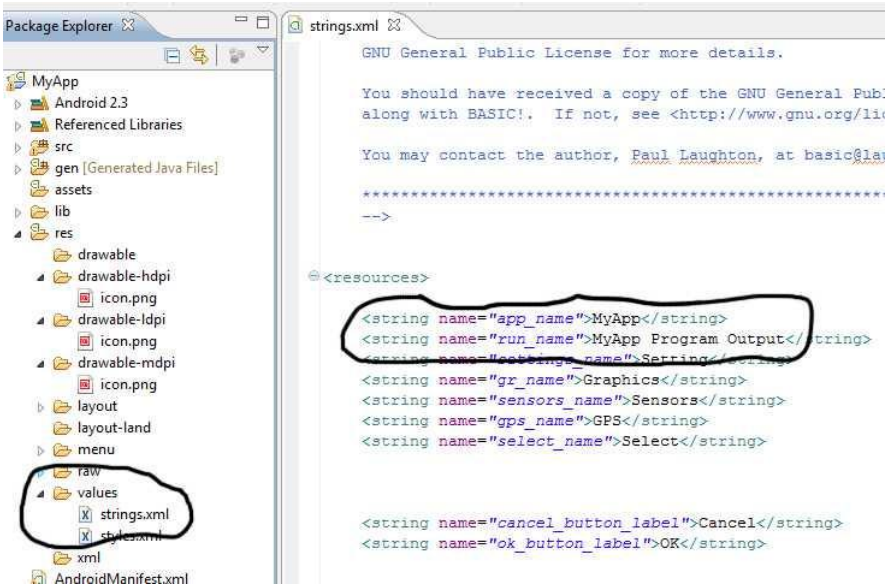
Outside of Eclipse, copy the icon file
In Eclipse, right click on the appropriate drawable- for the copied icon's size
Select Paste
Right click on the icon.png file and delete it.
Select the newly pasted icon and rename it to "icon.png" by selecting File -> Rename.

Yes, it is tedious work.

Naming The Application

Next we will define the name of the application and set the label that will appear on the text output screen.

Under res, open "values"
Double click on strings.xml to open it.
If you do not see what is shown below, click on the "string.xml" tab of the strings.xml window.



The application name is defined by the "app_name" parameter.

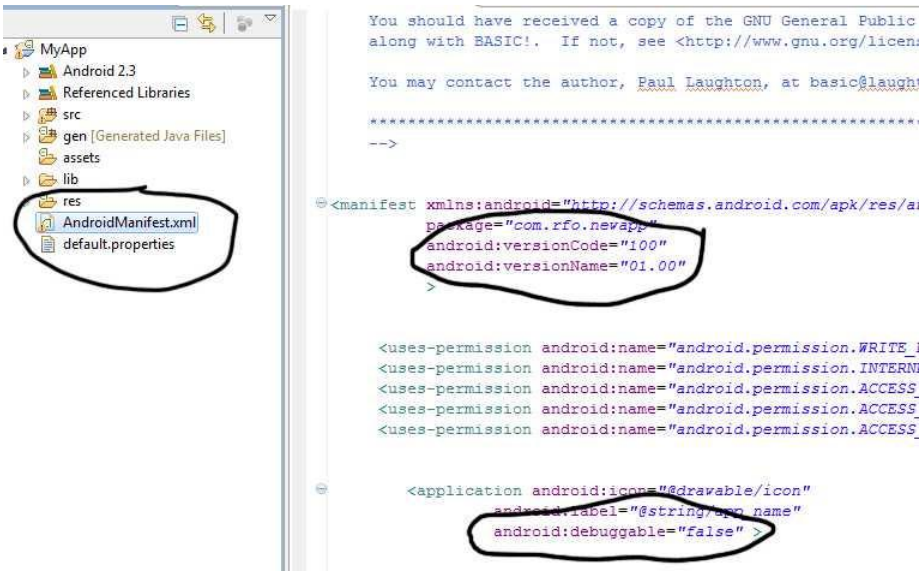
The title in the Text Output Screen Title Bar is defined by the "run_name" parameter.

Make your changes, click on the X in the tab to close and save the changes.

Setting The Version Number and 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.

Note: I have circled the android:debuggable parameter. You will need to change this to "true" if you need to do some serious debugging of the BASIC! code. In that case you have probably made changes beyond the scope of this tutorial. In any case, the parameter must be set to false when putting the application into the Android Market.

All the required changes have been made. Now it is time to test.

Warnings

Eclipse will show a bunch of warnings. Ignore them.

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="…….
```

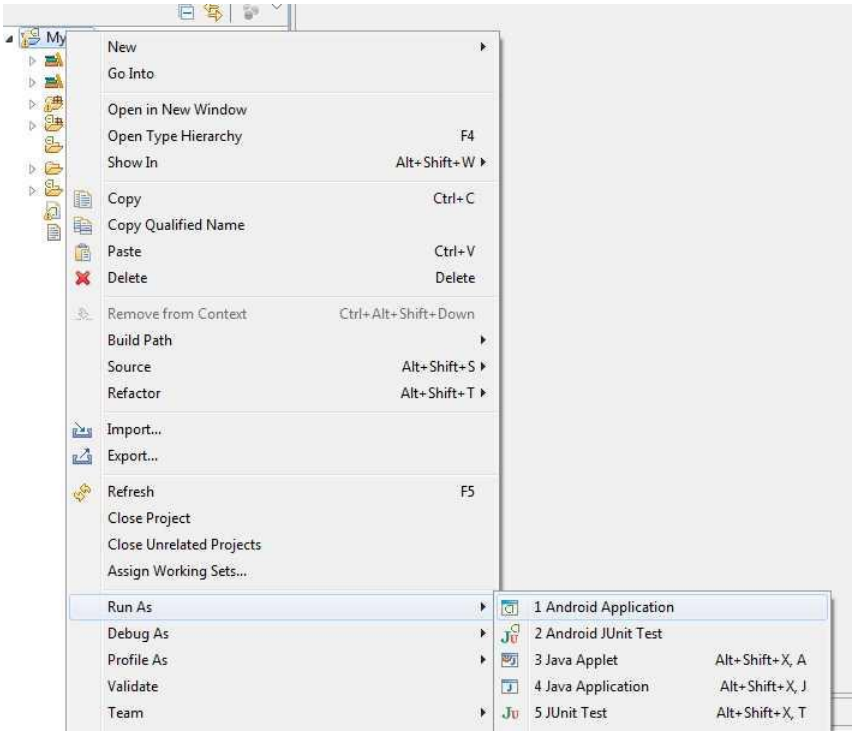
Look them over. If you feel that your APK does not need them then delete or comment them out.

Be sure to test your APK after doing this.

Compiling And Testing The Application

Make sure that you have the USB drivers for your Android device installed on your computer.
Connect the device to the computer using the USB cable.
On the device, open the Settings application.
Select Applications -> Development -> USB debugging.

In Eclipse, right click on MyApp, Select Run As - > Android Application

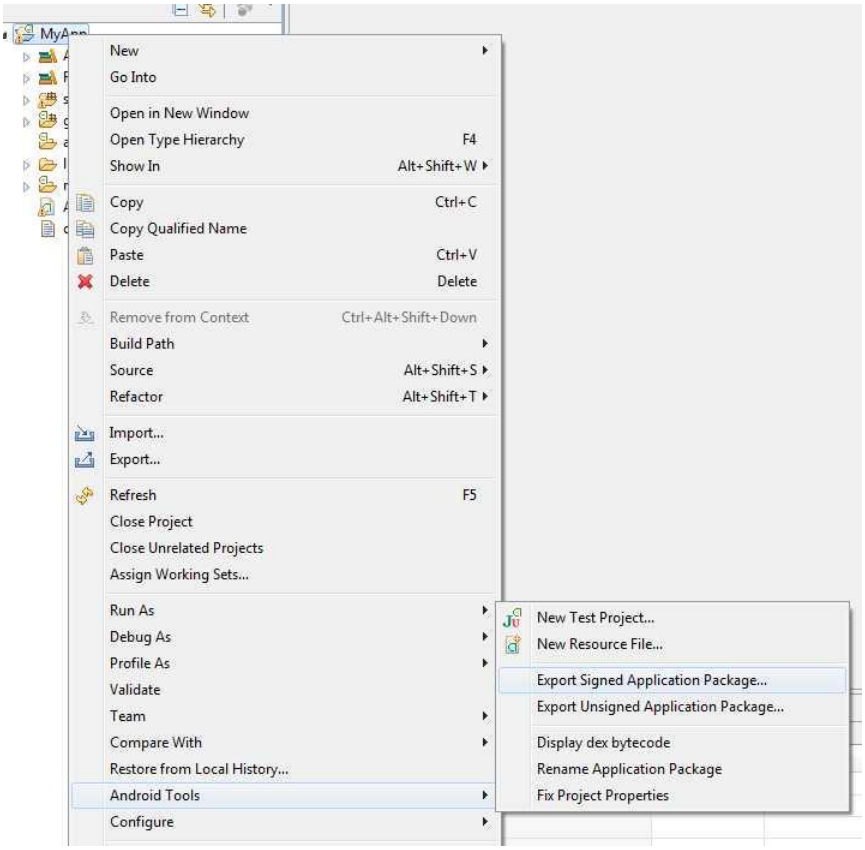


If all has gone well, your application should get compiled, downloaded to your device and be running.

Creating An .apk File

Start the process by right clicking on MyApp.

Select Android Tools - > Export Signed Application Package.



Follow the instructions to create a new key store.

I have never figured out what the Alias name is all about. Just give it some name.

If you are going to put the application in the Android Market, Google wants the validity duration to be 20 something years. I can't remember the exact number. I use 30 years. It is easy to remember.

Finished

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/).