DON'S BASIC LIBRARY

MANUAL

Version 2017-12-28

Table of Contents

1 Written by

Donald Hosford (Donzerme@Yahoo.com)
(Also SirDonzer on the rfobasic.freeforums.org)

2 Purpose

I wrote this Library of functions/subroutines to handle an addressable text display, with an onscreen keyboard. I wanted to try out some programs in some old magazines I have. Also I enjoyed the challenge of writing it.

3 Usage

Free for any use… If you do anything cool with it, let me know. Also let me know if you have any ideas, questions, or fixes.

4 Disclaimer (Mindless/Needless/Legal-sleeze/Crap/etc.)

This program library is Free to use, Free to distribute. ABSOLUTELY NO WARRANTY. USE AT OWN RISK.

5 Acknowledgments

Thanks guys!

Gcol:
Roy His Excellent Idea to simplify the colors! Why didn't I think of it!
Mougino Nice idea to reduce the color codes to a short function.
Me I added 11 more "web safe" colors. Added all the colors in a string.

Good ideas for the library: (mostly bundle usage, etc.)
Spike
aFox
Humpty

Help with the manual:
Spike Grammar. Subtle word choices. Really cool!
Robert A. Rioja Help with the manual. This does look much better than my simple text file!

6 Included files

DBLManual.pdf	This file.
DonsBasicLibrary.bas	The library include file.
IncKeyboard.bas	A simple typing keyboard include file.
F11_ Graphics_touchDBL.bas	The F11 sample program with an on-screen keyboard!

F11Keyboard.bas	The F11 sample program keyboard include file
DBLInputDemo.bas	Input Demo program.
DBLStartBlock.bas	Sample start program.

7 Information (about various things)

7.1 Version Number

The version number is simply the day I last changed something in that file. It is always listed at the top of the file, after the filename.

7.2 Command Naming

To keep my library commands from affecting any other Basic! commands, all functions start with the letters: "L_".

7.3 Bundle #1

This Library uses the "Bundle #1 Trick" to store it's data. (For those who don't know, the "Bundle #1 Trick": In Basic! Functions are supposed to have all data passed to them when they are called. It was discovered that Functions can access the Bundles. So Bundle #1 is can be used to pass much more info. See the DE RE Basic manual – Bundles.) The Library uses its own bundle. To avoid duplicating Bundle Keys, create any needed bundles before running L_librarySetup.

7.4 BMP Images

The Text Displays and the Keyboards are drawn on BMP images. Each screen or keyboard page adds a single graphical object to the display list. This keeps GR.Render times short. An 80 x 25 text display done with individual graphical text objects would add 2000 objects to the display list...

7.5 Scaling

This Library can scale the graphics for you. Scaling is not needed if only the Text Display/Keyboard is being used. During setup, it calculates how big everything should be, and draws it, on the fly. Every text display/keyboard will appear the same way on every device. If you are using graphics (circles, lines, etc.), scaling the screen is important. The scaling values are entered into the L_librarySetup command line. See L_LibrarySetup for usage.

8 Library Layout

The library functions are laid out in sections. Feel free to browse the library include file.

Text Display functions.
Onscreen Keyboard functions.
Graphics Functions.
Miscellaneous Functions.
Subroutines.

9 Installation

Copy the Library and keyboard include files into your source directory. Feel free to make a backup of the IncKeyboard.bas file if you want to modify it.

10 Using the library

To use, put the following commands at the start of your program:

To use a text screen only:

```
INCLUDE DonsBasicLibrary.bas
INCLUDE IncKeyboard.bas
!add any functions here.
L_LibrarySetup(1,0,1,0,0)
L_addscreen(80,25,10)
!Add any graphics here.
L_changescreen(1)
```

To use a text screen and keyboard:

```
INCLUDE DonsBasicLibrary.bas
INCLUDE IncKeyboard.bas
!add any functions here.
L_LibrarySetup(1,0,1,0,0)
!add as many screens as needed here.
L_addscreen(80,25,10)
!Add any graphics here.
L_keyboardsetup(225,9)
L_hidescreen(0)
L_changescreen(1)
```

To use the keyboard only:

```
INCLUDE DonsBasicLibrary.bas
INCLUDE IncKeyboard.bas
!add any functions here.
L_LibrarySetup(1,0,1,0,0)
!Add any graphics here.
L_keyboardsetup(225,9)
L_hidescreen(0)
```

The numbers in these commands can be changed. See the individual commands in the

command list. Remember these only start up the library. You have to add commands to write to the text display, retrieve key touches from the keyboard, etc.

Display List Order:
In RFO Basic! graphics draw order is important. If you need to draw graphics, put those commands between the L_addscreen, and L_keyboardsetup. The text screen(s) must be the first thing drawn, and the keyboard must be the last. Or the keyboard won't appear above the text screen/graphics.

Data:
Any program data, must appear AFTER the IncKeyboard include. And any data reads must be after the L_keyboardsetup command. The Keyboard reading function assumes the keyboard data is the first data in the program.

Bundles:
Any bundles that are created, should be created before L_librarySetup. This keeps the bundles for your program, and the library separate.

10.1 Scaling

The text screen and keyboard are automatically scaled to your display. Any size text screen / keyboard will appear the same way on everything that can run RFO Basic!. So scaling isn't needed if your program is only using the text screen/keyboard.

10.2 Text screen

It provides a variable size, addressable, multi-screen, text display!

10.2.1 Variable Size

You specify how many rows and columns you need for your data. L_addscreen figures out how big the text can be for the entire text screen to fit on the display. (You can enter crazy settings, but reading the resulting text screen is another matter...)

10.2.2 Addressable

Use L_Locate(x,y) to position your text. The upper left corner of the text display is 1,1. Old school programs can use L_LocPA(number) to Print @ a location on the text screen. (So go ahead and port those old programs!)

10.2.3 Multi-screen

Multiple text screens can be used at the same time. Each use of L_addscreen adds a new text screen. Very handy! Like putting your data on one screen, and the program help file on another. Switching between text screens is very fast. No waiting for the display redraw lots of text. Also the cursor position, Colors used, etc. for each screen are stored separately.

10.2.4 BMP Image

Only a single BMP image (graphical object) is used for each text screen. So little impact on Gr.render times.

10.2.5 Semigraphics

Supports three semigraphics modes! Early computers didn't have much, if any graphics. So they used graphical characters in the character set. The modes supported here, are semigraphic 4, semigraphic 6, and semigraphic 8. The semigraphic 6 mode is modeled after that used on the Radio Shack Model 1 computer. The 4 and 8 modes are modeled after that used on the Radio Shack Color Computer (Sort of...The color computer used 128 characters for semigraphics! It was semigraphics mode 4 repeated 8 times -- once in each foreground color!).

10.2.6 Text Screen Cons

It does have a few limitations:

The text on the screen, does not scroll!

No screen memory! Only the BMP Image exists. No fancy things like blinking text.

Ascii code zero can't be sent to the screen! Screws things up! (Seems to be a problem with RFO Basic?) Every ascii code zero is changed into a space (ascii code 32).

10.3 Onscreen Keyboard

Has two important features: Semi-transparent, and Hideable!

10.3.1 Semi-transparent

Mostly transparent, so you can see right through it! (Depending on the colors used.) See what you are typing! Read the results of that last input. A lot of the old programs were input driven. Enter something, it would process it, and display the results, and ask for more input. Porting these old programs over to RFO Basic! has the problem of not being able to read the results because of the Input popup, and onscreen keyboards. (I know rewriting the program is possible, but that seems to remove some of the charm of the old programs.)

10.3.2 Hidable

The keyboard has a Hide/show key. During L_Input, this hides the keyboard, without interrupting the input. When your ready, just tap the show key, and your keyboard is back.

10.3.3 101-text keys/12-control keys

Has most of the common keys I find on my PC keyboards. The included keyboard doesn't have every key, but you can always modify the keyboard file to include them. (Some will require more code...)

10.3.4 ASCII/Unicode

The library uses numbers to store the key values, so any ASCII/Unicode key can be added.

BMP Image:
Each "page" of a keyboard is drawn on a single transparent BMP image. Also little impact on graphics render times.

10.3.5 Keyboard Cons

It does have a limitation:
Only one keyboard include file can be loaded at a time. It will only use the first one loaded. A single keyboard file may contain the data for a single keyboard (the button grids, and it's button list. See the INCKEYBOARD.BAS file.

10.4 Colors

(Used by various Graphics setup and write functions.)

10.4.1 Common colors

Twenty different common colors. Colors may be referenced by a number 1 to 20, or using a "billion" code.

"Billion" code: store the decimal values red, green, blue with a one proceeding it. Zeros must be included. Billion codes allow any color with red/green/blue values to be used, and passed through the library functions as a single numeric value. The leading "1" allows the leading zeros to be kept. (Billion Code Example: Tan - Color 18 - Hexidecimal FFCC99 - Decimal 255,204,99 becomes 1255204099)

10.4.2 The Color Names and their Codes

In order from 1 to 20:
Black, Red, Lime, Blue, Yellow, Cyan, Magenta, Grey, White, Navy, Green, Teal, Maroon, Purple, Olive, Brown, Silver, Tan, Orange, Gold

	DECIMAL			HEXADECIMAL	NUMBER	NAME
"Billion" Code	RED	GREEN	BLUE	RedGreenBlue		
1000000000	0	0	0	000000	1	Black
1000000099	0	0	99	000099	10	Navy
1000000255	0	0	255	0000FF	4	Blue
1000099000	0	99	0	009900	11	Green
1000099099	0	99	99	009999	12	Teal
1000255000	0	255	0	00FF00	3	Lime
1000255255	0	255	255	00FFFF	6	Cyan
1099000000	99	0	0	990000	13	Maroon
1099000099	99	0	99	990090	14	Purple
1099099000	99	99	0	999900	15	Olive
1099099099	99	99	99	999999	8	Grey
1099033033	99	33	33	993333	16	Brown
1204204204	204	204	204	CCCCCC	17	Silver
1255204099	255	204	99	FFCC99	18	Tan
1255000000	255	0	0	FF0000	2	Red
1255000255	255	0	255	FF00FF	7	Magenta
1255099000	255	99	0	FF9900	19	Orange
1255204000	255	204	0	FFCC00	20	Gold
1255255000	255	255	0	FFFF00	5	Yellow
1255255255	255	255	255	FFFFFF	9	White

11 List of Commands (Functions/Subroutines)

11.1 L_Addscreen(DisplayMX,DisplayMY,Displaybackground, Scale1,Scale2)

Adds a new text screen, to the end of the screen list. The first screen added is number 1, the next is number 2, and so on. Any number may be added up to the limits of your device’s memory.

DisplayMX, and DisplayMY define the number of columns, and rows of the display.

Displaybackground is the background color for the new text screen.

Scale1/Scale2 are the values to scale the screen.

11.2 L_ChangeScreen(NewScreen)

Changes the current screen to the "NewScreen". Hides the previous one.

11.3 L_Clear()

Clears the current text screen to the current background color.

11.4 L_ClearABox(TCX,TCY,Width,Hight,Color)

Clears an area of the text screen to the assigned color. All numbers are in characters. (See L_Locate)

Width/Hight defines how big the box is, in characters.

TCX/TCY is the position of the upper left corner of the box on the screen.

Color is the desired color to change the box to.

11.5 L_CurrentKeyboard(Keyboard)

Switches the current keyboard to the specified keyboard number. Hides the previous keyboard. Any calls to L_Whatkey uses the last set keyboard.

11.6 L_Drawbuttons()

Internal. Called by L_Keyboardsetup. Draws the onscreen keyboard contained in the included keyboard file.

11.7 Getcursor:

Subroutine. Returns the current text screen cursor location. Sets TCX, and TCY variables.

11.8 GetLastKey:

Subroutine. Gets the last key touched on the onscreen keyboard. Sets Type$, and Keytouch.

Keytouch: Numeric. Ascii/Unicode character.

Type$: String. Shows weather keytouch is a control key or text key.

Interpreting the results. If type$ is a "1", then a text key. If it is a "0", then a control key. Keytouch contains a number. Control key number is simply the position of the key from the keyboard’s Button List. Text key is the ASCII/Unicode value.

11.9 Gettouch:

Retrieves touches from the graphic screen. Sets:

T1X/T1Y - location of the first touch start, in pixels.

T2X/T2Y - location of the first touch stop, in pixels.

Does not intercept the second touch. If scaling is used, the output of Gettouch: is already scaled for you.

11.10 L_Gcol(Alpha,Color,Fill)

Internal. Graphics. Sets the color. (See "Colors".) Color number can be 1-20, or a “Billion” code. Billion codes are 10 digits long. The leading 1 only holds the number at 10 digits. (example: 1255204099 = Tan. See color list.)

11.11 Grabakey:

Subroutine. This functions exactly like Inkey$, only for the onscreen keyboard. Retrieves a single keypress. Returns Type$ and Keytouch, just like Getlastkey. Must use L_showkeyboard(Keyboard Number) first. It is just Gettouch:, L_Whatkey, and GetLastKey: combined into a single subroutine. Simplifies things a little.

11.12 Grabaresult:

Subroutine. Retrieves the result of L_input(Text$). Stored in Result$.

11.13 L_Hidekeyboard(Keyboard)

Hides the keyboard specified.

11.14 L_Hidescreen(Screen)

Hides the specified text screen.

11.15 L_Input(TEXT$)

Works like "input". Gets keystrokes from the onscreen keyboard. Each keystroke is written to the current text screen. The result must be retrieved from the bundle. See the Grabaresult subroutine.

11.16 L_KeyboardSetup(KeyboardAlpha,KeyboardColor)

Sets up the onscreen keyboard. This reads the keyboard data, stores it in the bundle, and draws the keyboard pages.

KeyboardAlpha is the "alpha" of the drawn keyboard. (ie: How transparent it is.)

KeyboardColor is the color used to draw the keyboard. (see Colors.)

11.17 L_LibrarySetup(DisplayBoarder,DisplayORI,DisplayStatus,Scale1,Scale2)

Must be the first library command run. Sets up the library. Opens the graphic screen, sets up the bundle, sets the scale values (if desired). Example: L_LibrarySetup(1,0,1,800,480)

DisplayBoarder: Basic's graphic screen color. Only the edges of the screen will show. (See colors.)

DisplayORI: Sets the screen orientation. A "1" (one) for Portrait orientation, and a "0" (zero) for Landscape orientation. (See De Re Basic Manual – Gr.Orientation)

DisplayStatus: Shows the device's status bar at the top edge of the screen. Set to "1" (one) to show the statusbar, and "0" (zero) to hide the statusbar.

Scale1/Scale2: Sets the scale values to scale all graphics. The order of the two scale values does not matter. Make sure to set DisplayORI to the correct value. To scale the output, set Scale1/Scale2 to the screen size(in pixels) that you want. (Like 800 by 480.) To turn off scaling, set both values to zero. This turns off Basic's Scaling command. The library uses either the device's actual screen size, or the specified scale values, to calculate the sizes of

the screen/keyboard.

11.18 L_LoadButtonData()

Internal. Called by L_keyboardsetup. Reads the keyboard data, stores it in the bundle.

11.19 L_Locate(TCX,TCY)

Sets the location of the text cursor on the current text screen. Any off screen coordinates will be "wrapped" to the opposite edge. Coordinates are in character positions. (not pixels.) The upper left corner of the screen is location 1,1. The lower right corner is location DisplayMX, DisplayMY.

11.20 L_LocPA(PrintAt)

Allows the use of old "print at" coordinates. Converts it to a L_locate position, and stores it. The Radio Shack Model I/Color computers used this form of locate. The upper left corner was location 0, the one to the right was location 1, etc.

11.21 L_NumberClip$(Number,Number$)

Returns a number in a string, without the decimal.
Example: A$=L_numberclip$(5,A$) A$ have a " 5" in it.

11.22 L_OpenGR(Alpha,color,Status,Orientation)

Internal. Called by L_LibrarySetup. Opens the graphic screen, sets the status and the device orientation.

11.23 L_Setcolors(foreground,background)

Sets the foreground/background colors for text writing. Color values may be 1-20 or Billion Codes.

11.24 L_Showkeyboard(Keyboard)

Shows the specified keyboard.

11.25 L_Tab$(Tab)

Returns a string "tab" characters long. Example: A$=L_tab$(35) Returns a string 35 spaces long.

11.26 L_Textmode(TextMode)

Switches the text mode used in the next L_write. The selected TextMode stays active until changed by another L_textmode command.

Allowed modes:
1 - Normal text
4 - Semigraphics 4 mode
6 - Semigraphics 6 mode
8 - Semigraphics 8 mode

11.27 L_WhatKey(TouchX1,TouchY1)

Figures what key was pressed (if any), from the touch coordinates provided. Use GetLastKey: to retrieve the keypress.

11.28 L_Winfill(TCX,TCY,Width,Hight,Text$)

Puts Text$ on the current text screen in a box defined by Width and Hight. Its upper left corner will be at TCX/TCY. If Text$ is longer than Width x Hight, the rest of the string will be ignored.

11.29 L_Write (Text$)

Writes text$ to the current text screen, starting at the current cursor location. Numbers must be converted to strings first. Always assumes concatenation. (Concatenation example: Print "sail"; "boat" -- prints "sailboat") The following cursor control codes may be included in a text string: (must be in text mode 1)

ASCII	action	Meaning
1	"Home"	Moves the cursor to location 1,1.
2	"Cursor Up"	Moves the cursor Up one line (towards the top of the screen).
3	"Cursor Down"	Moves the cursor Down one line.
4	"Cursor Left"	Moves the cursor Left one position.
5	"Cursor Right"	Moves the cursor Right one position.
8	"Backspace"	Moves the cursor Right one position, and erases the character there.
13	"Enter"	Moves the cursor to the beginning of the next line down.

Semigraphics:
Simple graphic block characters. (Used by early home computers in place of graphics, because of too little memory.) These characters are not a font, they exist only on the library's text display(s).

To use: Set L_textmode to 4, 6 or 8. Then use L_Write to put characters on the text display.

Semigraphics mode 4 uses the four lowest valued bits of each character. The other four bits

are ignored. Semigraphics mode 6 uses the six lowest valued bits of each character. The semigraphics mode 8 uses all eight bits of each character.

With eight bit characters, examining the bit patterns used, semigraphics mode 4 is repeated 16 times, and semigraphics mode 6 is repeated four times. Because L_write ignores the unused bits, these two modes don't have to use only the first 16 or 64 character codes. (On the Radioshack Model 1, semigraphics 6 mode uses the ascii codes of the third set - 128 to 191. So no translation of values are needed.)

12 Adding a screen

The text display system uses "screens". Each screen is a BMP image. To add a screen, use the L_addscreen(Width,Hight,Color). The Width is in columns of text. The Hight is in rows of text. Color is a color number. The command must be used once for each screen to be added. (Example: Adding two 80 x 25 screens, requires the command L_addscreen(80,25,10) to be done twice.)

(Example: L_addscreen(80,25,10) creates a display 80 columns wide, by 25 rows high, with a navy background color.)

The screens are numbered. The first created is number one, the second number 2, and so on. Each screen created may be different sizes.

These screens are "addressable". The upper left corner is 1,1. Use the command: L_locate(Column,Row) to position the cursor onto the screen. The text cursor will be re-positioned at the end of whatever was written.

Use the command L_write("some text") to write text into the current screen. String variables may be used also. (L_write(A$)) Any numbers to be writen need to be converted to strings first. (A$=str$(45))

Use the command L_ChangeScreen(NewScreen) to change to a new screen. This will hide the old screen, and show the new one.

13 Reading the keyboard

Reading the keyboard is simple. If using the text display, just use the following code:

```
L_input(Result$)
Gosub Getresults
```

Returns the typed input in Result$.

To write your own input, use the following code:

```
Gosub GetTouch
L_whatkey(t1x,t1y)
Gosub GetLastKey
```

This returns Type$, and Keytouch.
If Type$="1", a text character was touched. Then KeyTouch contains the ASCII/Unicode value of the key touched.
If Type$="0", then a control key was touched. KeyTouch contains the key number (from the button list) for control keys.

I process control keys like this:

```
 IF Type$="0"
   if keytouch=1
         !control key #1 code
        endif

   etc…

 endif
```

14 Modifying the keyboard

Adding your own keys to the keyboard is easy. Refer to the Sample Keyboard I have included below. (section 14.4) It is the current INCKeyboard.bas file.

14.1 The keyboard file

The file is filled with data statements. These are divided into three kinds: Keyboard info, Button Grids, and the Button List. Keyboard info describes the keyboard. How many pages there are, the largest Button Grid, etc. The Button Grids show how many "pages" of keyboards there are, and what buttons are where on the keyboard. The Button List is a simple list of all the keys, on all the pages of the keyboard.

On the Button Grids, a zero indicates there is no button there. A space will be left on the onscreen keyboard. Numbers indicate which button from the Button List will appear there. A 1 is the first button on the list, 2 is the second, and so on. Some numbers are repeated, like key 5 (shift). It occupies two adjacent positions. This shows that those two positions will appear as a single button on the onscreen keyboard. (With the button name centered.)

On the Button List, each row has the data for five buttons on it. each one is laid out: TypeName / Asciicode / keyname. Type is the key type, a one or a zero. Ones signify text keys, and zeros signify control keys. Name is the key name. This is what appears on the key. If it is a space (" "), then the ascii code will be used instead. Ascii code can be ascii or unicode. Keyname is just there for human readablity. (So I can easliy find a key in the Button List!)

Adding keys: Just add a key(s) to the end of the Button List. Increment the "number of unique Button names" to the correct value. Follow the key format. Add the Key number to a button grid. Feel free to rename the resulting keyboard file. Be sure to include the new keyboard in your programs. Probably should make a backup of the main keyboard file first…

14.2 Building A New Keyboard

Keyboards can be made for any special purposes. Just follow the sample file below.

A) Purpose. Deside on the purpose of your keyboard. (Typing? Game controls? Something else?)
B) Keyboard orientation. Which way will the screen be?
C) How many "pages" in the keyboard? First data item. Number of grids.
D) Largest Grid. List the largest grid here, after deciding how big they each are. (they don't have to be the same size.)
E) First Button Grid. First place the grid size. Number of columns wide, then number of rows high.
F) Layout the Button Grids. Layout the grid of zeros to match the columns/rows. They don't have to be arranged like this, but it does make the keyboard file easier to fix. I planned my keyboard on graph paper first, then typed it in. Repeat for each Button Grid.
G) Single character/multiple character. A left over from an earlier keyboard layout. For now just put 1,1 for each. I may fix this later...
H) Button List. Make a list of all the buttons that will be on the keyboard. Follow the data layout discussed above. I like to list them in rows of five buttons.
I) Number of unique buttons. Count up the total number of buttons on the list. Put it at the top of the button list.
J) Fill the Grids. Place the button numbers in the grids. The numbers are simply the key's position in the button list. The first key is number 1, the second key is number 2, and so on. To make larger buttons, place the same button list number in adjacent positions. (Square or Rectangle shaped buttons are best.)
K) Save the file.
L) Make sure to INCLUDE your keyboard file, in place of the INCKeyboard.bas file.
M) Done!

Notes:
If you want to use the library's L_Input, or GrabAKey functions, copy the first twelve keys of INCKeyboard.bas file to the top of your Button List. They must be in the same order, in the same position! The L_Input function expects them that way. (Example: It doesn't know that Enter is key 3...only that it must process key 3 as an enter.)

Some characters can't appear in the data statements. Use ascii/unicode codes instead. Quote is one of these.

14.3 Multiple keyboards

The Library can only load, and read the first single keyboard include file. The file can contain a

single set of keyboard data. If multiple keyboards are desired, all desired keyboards must be combined into a single file. That is, all of the button grids must be together, all of the Button Lists must be together. The number/size of the Button Grids must be updated. The total number of Buttons on the Button List, must be updated. The button numbers on the added Button Grids must be renumbered to match the new Button List positions.

14.4 Keyboard Sample

Included for reference purposes.

```
!Typing Keyboard data
!This is the second version of the "typing" keyboard.
!Version: 11-8-2017
!---------------------------

!Button Grids
!number of keyboards
READ.DATA 3
!LargestGrid X,Y -- number of columns, number of rows
READ.DATA 14,7

!#1 -- Unshifted
READ.DATA 14,7
READ.DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
READ.DATA 77, 30, 31, 32, 33, 34, 35, 36, 37, 38, 29, 26,  6,  6
read.data  4, 94,100, 82, 95, 97,102, 98, 86, 92, 93, 42, 73,111
READ.DATA  2,  2, 78, 96, 81, 83, 84, 85, 87, 88, 89, 40,  3,  3
READ.DATA  5,  5,103,101, 80, 99, 79, 91, 90, 25, 27,  7, 28,112
READ.DATA 12,  1,  1, 72, 74, 13, 13, 13, 13, 20,  8,  9, 10,113
READ.DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
!Button size: Single Character
READ.DATA 1,1
!Button Size: Multiple characters
READ.DATA 1,1

!#2 -- Shifted
READ.DATA 14,7
READ.DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
READ.DATA 107, 14, 45, 16, 17, 18, 75, 19, 23, 21, 22, 76,  6,  6
READ.DATA  4, 62, 68, 50, 63, 65, 70, 66, 54, 60, 61, 24,105,108
READ.DATA  2,  2, 46, 64, 49, 51, 52, 53, 55, 56, 57, 39,  3,  3
READ.DATA  5,  5, 71, 69, 48, 67, 47, 59, 58, 41, 43,  7, 44,109
READ.DATA 12,  1,  1,104,106, 13, 13, 13, 13, 15,  8,  9, 10,110
READ.DATA  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
!Button size: Single Character
READ.DATA 1,1
!Button Size: Multiple characters
```

```
READ.DATA 1,1

!#3 -- show
READ.DATA 14,7
READ.DATA   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
READ.DATA   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
READ.DATA   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
READ.DATA   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
READ.DATA   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
READ.DATA  11,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
READ.DATA   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
!Button size: Single Character
READ.DATA 1,1
!Button Size: Multiple characters
READ.DATA 1,1

!Button List
!number of unique button names
READ.DATA 113
!format: TypeName/asciicode/keyname
READ.DATA "0Menu/0/ ","0CapLock/0/ ","0Enter/13/ ","0Tab/0/ ","0Shift/0/ "
READ.DATA "0Back/8/ ","0 /9651/Up","0 /9665/Left","0 /9661/Down","0 /9655/Right"
READ.DATA "0Show/0/ ","0Hide/0/ ","1 /32/space","1 /33/!","1 /34/quote"
READ.DATA "1 /35/#","1 /36/$","1 /37/percent","1 /38/ampersand","1 /39/Apost"

READ.DATA "1 /40/(","1 /41/)","1 /42/*","1 /43/plus","1 /44/comma"
READ.DATA "1 /45/-","1 /46/period","1 /47/Fslash","1 /48/0","1 /49/1"
READ.DATA "1 /50/2","1 /51/3","1 /52/4","1 /53/5","1 /54/6"
READ.DATA "1 /55/7","1 /56/8","1 /57/9","1 /58/:","1 /59/;"

READ.DATA "1 /60/<","1 /61/=","1 /62/>","1 /63/?","1 /64/@"
READ.DATA "1 /65/A","1 /66/B","1 /67/C","1 /68/D","1 /69/E"
READ.DATA "1 /70/F","1 /71/G","1 /72/H","1 /73/I","1 /74/J"
READ.DATA "1 /75/K","1 /76/L","1 /77/M","1 /78/N","1 /79/O"

READ.DATA "1 /80/P","1 /81/Q","1 /82/R","1 /83/S","1 /84/T"
READ.DATA "1 /85/U","1 /86/V","1 /87/W","1 /88/X","1 /89/Y"
READ.DATA "1 /90/Z","1 /91/[","1 /92/Bslash","1 /93/]","1 /94/^"
READ.DATA "1 /95/underscore","1 /96/`","1 /97/a","1 /98/b","1 /99/c"

READ.DATA "1 /100/d","1 /101/e","1 /102/f","1 /103/g","1 /104/h"
READ.DATA "1 /105/i","1 /106/j","1 /107/k","1 /108/l","1 /109/m"
READ.DATA "1 /110/n","1 /111/o","1 /112/p","1 /113/q","1 /114/r"
READ.DATA "1 /115/s","1 /116/t","1 /117/u","1 /118/v","1 /119/w"

READ.DATA "1 /120/x","1 /121/y","1 /122/z","1 /123/{","1 /124/|"
READ.DATA "1 /125/}","1 /126/~","1 /162/cent","1 /166/BrknVBar","1 /8230/dotdotdot"
```

read.data "1 /8226/bullet","1 /175/overline","1 /176/Degree"

!end.

15 Glossary

TERM	WHERE USED	DEFINITION
Alpha	L_Gcol L_OpenGR	How transparent something is.
Background	L_Setcolors	Background color setting for text writing.
Color	L_Gcol L_ClearAbox	Color codes, see "Color" above.
Displaybackground	L_Addscreen	Background color code for the text screen.
DisplayBoarder	L_LibrarySetup	Color code for the graphic screen color.
DisplayMX	L_Addscreen	Number of Columns wide the text screen is.
DisplayMY	L_Addscreen	Number of Rows high the text screen is.
DisplayORI	L_LibrarySetup	Orientation of the graphic screen. Portrait or Landscape. See De Re Basic manual – Gr.Orientation for values.
DisplayStatus	L_LibrarySetup	Sets weather to show or hide the device's status bar at the top edge of the screen.
Fill	L_Gcol	Color setting. To fill the text written or not.
Foreground	L_Setcolors	Foreground color setting for text writing.
Height	L_ClearABox L_WinFill	Total number of rows high the area is.
Keyboard	L_Currentscreen L_HideKeyboard L_ShowKeyboard	Number of the current keyboard.
KeyboardAlpha	L_KeyboardSetup	Alpha setting for the drawn keyboard.
KeyboardColor	L_KeyboardSetup	Color code of the drawn keyboard.
Keytouch	GetLastKey GrabAKey	Contains the control key number or an ASCII/Unicode value.
NewScreen	L_ChangeScreen	Number of the screen to change to.
Number	L_Numberclip$	Number to be processed.
Number$	L_Numberclip$	Result string.
Orientation	L_OpenGR	Orientation of the graphic screen.

TERM	WHERE USED	DEFINITION
PrintAt	L_LocPA	Early form of Locate used on the Radio Shack Model 1 Color Computer. Upper left corner was 0. The next position to the right was 1, and so on.
Result$	Grabaresult	Contains the results of L_Input.
Scale1	L_LibrarySetup	Device Independent Screen size. Example: Width, in pixels. Scale1 and Scale2 are interchangable.
Scale2	L_LibrarySetup	Device Independent Screen size. Example: Height, in pixels. Scale1 and Scale2 are interchangable.
Screen	L_hidescreen	Number of the text screen.
Status	L_OpenGR	Flag to turn on the status bar.
T1X/T1Y	Gettouch	Contains the location the user first touched the screen.
T2X/T2Y	Gettouch	Contains the location the user stopped touching the screen.
TCX	L_ClearABox L_Locate L_WinFill	Text Cursor X coordinate.
TCY	L_ClearABox L_Locate L_WinFill	Text Cursor Y coordinate.
Text$	L_Input L_Write L_Winfill	Text to be processed/written.
TextMode	L_TextMode	What text mode to use. Valid modes: 1, 4,6,8
TouchX1	L_WhatKey	Touch screen X coordinate, in pixels.
TouchY1	L_WhatKey	Touch screen X coordinate, in pixels.
Type$	GetLastKey GrabAKey	Show if keytouch code is a typeable key ("1"), or a control key ("0" - zero).
Width	L_ClearABox L_WinFill	Total number of Columns wide the area is.

16 Manual History

This manual was originally written by Don as a text file. It was then edited as a LibreOffice (.odt) file by Robert A. Rioja (robrioja@gmail.com). The .odt file was then exported as a PDF (.pdf) file.

Wowzers! This manual really look cool now! – Don. Thanks Robert!