Basic! User Manual

Originally published as De_Re_Basic!

Original Author Paul Laughton, 2011

Edited by Robert A. Rioja

robrioja@gmail.com

http://www.RvAdList.com

Version 2023-04-13

Table of Contents

1 Introduction

1.1 About the original title, De Re BASIC!

"De Re" is Latin for "of the thing" or "about". Therefore, “De Re Basic!” means “About Basic!”.

1.2 About the Cover Art

Thanks to BASIC! collaborator Nicolas Mougin. The images are screenshots from real BASIC! programs available from the Google Play™ store, or from the excellent collection of shared BASIC! programs available at the Basic! forum.

1.3 Credits

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

Thanks also to Mike Leavitt of Lansdowne, VA, USA, for his many contributions and long-time support.

1.4 Documentation

This document, ***Basic! User Manual***, was developed from the original ***De_Re_BASIC!*** document. It is a companion to the ***Basic! Language Reference*** also developed from the original ***De_Re_BASIC!*** document.

Both the ***Basic! User Manual*** and the ***Basic! Language Reference*** were edited, and are maintained, by Robert A. Rioja.

2 Permissions

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

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

3 Editor

3.1 Editing the Program

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

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

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

Some Android devices are shipped with "Settings/Developer Option/Destroy Activities" checked and/or "Settings/Energy/Quick Restart" checked. Both of these setting create problems with loading files into the Editor. It appears as if you have gone through the process of loading the file but nothing appears in the editor. The solution to the problem is to uncheck both of these options. Even better, completely turn off Developer Options unless you know that you have a legitimate development need.

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

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

3.2 Multiple Commands on a Line

More than one BASIC! source code statement may be written on one physical line. Separate commands with a colon character ":". For example, the following line uses three separate commands to initialize some variables:

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

Note that two commands, **Sensors.open** and **SQL.update**, use the colon as a sub-parameter separator. If you use multiple-command lines, be careful when using these two commands.

3.3 Line Continuation

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

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

For example, the code line:

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

could be written as:

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

3.4 # - Format Line

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

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

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

4 Menus

Press the MENU key or touch the Menu icon to access the following menus. On some versions of Android, you will not see all of the menu options. Instead, you will see the first five options and a **More** options. Select the **More** option to see all of the options listed.

4.1 Run

Run 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 the offending line will be shown as selected in the editor.

4.2 Load

Load a program file into the editor. The first time you open BASIC! after installing it, if you select **Load** it displays the sample programs in the directory **rfo-basic/source/Sample_Programs**. (See **Paths Explained**, later in this manual.) Otherwise, when you open BASIC! and select **Load** it starts in the default source directory **rfo-basic/source**. Program files must have the extension **.bas**.

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

The "BASIC! Load File" screen shows the path to your current directory followed by sorted lists of the subdirectories and program files in it. Directories are denoted by the **(d)** appended to the name. BASIC! programs are shown with the **.bas** extension. If there are files in the directory that do not have the **.bas** extension, they do not appear in the list.

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

You can navigate to any directory on your device for which you have **read** permission.

- Tap on a directory to display its contents.
- Tap on the ".." at the top of list to move up one directory level. The tap has no effect if the current directory is the device root directory **"/"**.

You can exit the **Load** option without loading a program by tapping the BACK key.

BASIC! remembers the path to the directory you are in when you load a program. Next time you select **Load**, it starts in that directory. If you select **Save** or **Save and Run**, the file is saved in the remembered directory (unless it is Sample_Programs).

4.3 Load and Run

Selecting this option is exactly the same as first selecting **Load** and then selecting **Run**. The selected program is loaded into the Editor and is run immediately.

4.4 Save

Saves the program currently in the editor.

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

If you do not enter a file name, BASIC! uses a default filename, **default.bas**.

BASIC! remembers the path to the directory you were in when you last loaded or saved a program. When you **Save**, the file name you type is saved in the remembered directory. If the name you type includes subdirectories, BASIC! remembers the new path. The name you type can include **"../"**. Be careful if you are using a soft keyboard, as it may automatically insert spaces that you don't want.

You cannot save programs in the sample program directory **source/Sample_Programs**. If you **Load** a program from **source/Sample_Programs**, change it, and **Save** it, the program is saved in **source**.

You can exit **Save** without saving a file by tapping the BACK key.

4.5 Save and Run

Selecting this option is a fast way to save and then run. Any changes you have made are saved, overwriting your file, and your program is run immediately. A brief popup notifies you that your file has been changed. If the program you are editing has no name (not previously loaded or saved), the Editor will ask you what name to use.

4.6 Clear

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

4.7 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.

4.7.1 NEXT Button

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

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

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

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

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

4.7.2 REPLACE Button

If **Next** has found and selected some text then that text is replaced by the contents of the **Replace With**

field.

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

4.7.3 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.

4.7.4 DONE Button

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

4.7.5 BACK Key

Returns to the Editor with the original text unchanged. All changes made during the Search will be undone. Think of the BACK key as UNDO ALL.

4.8 Format

Format the program currently in the Editor. The keywords are capitalized. Program lines are indented as appropriate for the program structure. Left- and right-double quotation marks (“ and ”) are replaced by simple ASCII quotation marks (").

When copying program text from the Forum or another web site, "non-breaking space" characters, designated ** ** in HTML, may be inserted into the program text. Except when they are enclosed in quoted strings, **Format** converts these characters to simple ASCII spaces.

4.9 Delete

Delete files and directories. The command should be used for maintaining files and directories that are used in BASIC! but it can also be used to delete any file or directory on the SD card for which you have the required permissions. **Delete** starts in the **rfo-basic** directory.

Tapping **Delete** presents the "BASIC! Delete File" screen. The screen shows the path to your current directory followed by sorted lists of the directories and files in it. Directories are marked with **(d)** appended to the name and appear at the top of the list.

Tapping a file name displays the "Confirm Delete" dialog box. Tap the **Delete** button to delete the file. Tap the **No** button to dismiss the dialog box and not delete the file.

Tapping a directory name displays the contents of the directory. If the directory is empty the "Confirm Delete" dialog box is shown. Tap the **Delete** button to delete the directory. Tap the **No** button to dismiss the dialog box and not delete the directory.

Tap the **".."** at the top of the screen to move up one directory level. Tapping the **".."** has no effect if you are in the root directory **"/"**.

Exit **Delete** by tapping the BACK key.

4.10 Preferences

4.10.1 Screen Colors

Opens a sub-menu with options for setting the colors of the various screens in BASIC!

4.10.1.1 Color Scheme

Sets the color scheme of the screens. The schemes are identified by their appearance with the default colors. Choose one of the following:

- Black Text On White Screen
- White Text On Black Screen
- White Text On Blue Screen

4.10.1.2 Custom Colors

Check the box to override the Color Scheme setting, allowing you to set your own colors. You can set the following options:

- Text Color
- Background Color
- Line Color
- Highlight Color

Each color is specified as a single number of 8 hexadecimal characters: four fields of two characters each for Alpha (opacity), Red, Green, and Blue components.

4.10.2 Console Settings

Opens a sub-menu with options for settings of the Console and various others screens in BASIC!

4.10.2.1 Font Size

Sets the font size to be used with the various screens in BASIC! as follows:

- Small
- Medium
- Large

4.10.2.2 Typeface

Choose the typeface to be used on the Output Console and some other screens:

- Monospace
- Sans Serif
- Serif

4.10.2.3 Console Menu

Check the box if the Menu should be visible in the Output Console and TGet screen.

4.10.2.4 Console Lines

Check the box if the text lines in the Output Console should be underlined.

4.10.2.5 Empty Console Color

Choose the background color of the part of the Output Console that has not yet been written. It can match the background color of the text or the color of the lines separating text lines:

- Use text background color

- Use separator line color

This setting also applies to the **Select** (but not **Dialog.select**) command.

4.10.3 Editor Settings

Opens a sub-menu with options for setting properties and features of the Program Editor.

4.10.3.1 Editor Lines

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

4.10.3.2 Editor Line Wrap

Check the box if long text lines in the Editor should wrap at the edge of the screen. If unchecked, long lines are not wrapped, and the Editor screen may be scrolled horizontally.

4.10.3.3 Editor AutoIndent

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

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

4.10.4 Menu Items On Action Bar

Opens a sub-menu with options for moving some of the Editor menu items to the Action Bar, if there is room for them there. You can select as many as you like, but the number of items moved depends on the device and orientation. These options have no effect on Android devices before Honeycomb (3.0).

4.10.4.1 RUN on action bar

If checked, the Editor will attempt to move the RUN item from the Menu to the Action Bar.

4.10.4.2 LOAD on action bar

If checked, the Editor will attempt to move the LOAD item from the Menu to the Action Bar.

4.10.4.3 SAVE on action bar

If checked, the Editor will attempt to move the SAVE item from the Menu to the Action Bar.

4.10.4.4 CLEAR on action bar

If checked, the Editor will attempt to move the CLEAR item from the Menu to the Action Bar.

4.10.4.5 SEARCH on action bar

If checked, the Editor will attempt to move the SEARCH item from the Menu to the Action Bar.

4.10.4.6 EXIT on action bar

If checked, the Editor will attempt to move the EXIT item from the Menu to the Action Bar.

4.10.5 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:

- Variable By Sensors
- Fixed Landscape
- Fixed Reverse Landscape

- Fixed Portrait
- Fixed Reverse Portrait

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

4.10.6 Graphic acceleration

Check this box to enable GPU-assisted graphics acceleration on devices since 3.0 (Honeycomb) that support it. It is disabled by default. If you enable this option, test your program carefully. Hardware acceleration can make some of BASIC!'s graphical operations fail.

4.10.7 Base Drive

Some Android devices have several external storage devices (and some have no physical external storage devices). BASIC! will use the system-suggested device as its base drive. The base drive is the device where the BASIC! "rfo-basic" directory (base directory) is located. The base directory is where BASIC!'s programs and data are stored. See the "Files and Paths" chapter in the ***Basic Language Reference*** manual.

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

In this manual, <pref base drive> means the base drive you selected when you set the Base Drive here.

Note: If you have created a Launcher Shortcut (see chapter 17 Launcher Shortcut Tutorial) with files in one base directory but try to execute that shortcut while using a different base directory, the shortcut will fail to execute.

4.11 Commands

The Commands command presents the list of the BASIC! commands and functions found in chapter 15 Command List of this document.

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

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

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

4.12 About

The About option displays the version of BASIC! that you are using, followed by a set of buttons that connect you to various websites with information about BASIC!. Make sure that you have a connection to the Internet before selecting one of the About buttons.

4.13 Exit

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

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

5 Running a Program

5.1 Run

Selecting Run from the Editor's menu starts the program running. However, if the source in the Editor has been changed, then the Save dialog will be displayed. You may choose to save the changed source or continue without saving.

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

- The program prints something.
- The **END** statement is executed.
- You are in Echo mode.
- There is a run-time error.

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

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

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

If the program ended with a run-time error, the line where the error occurred will be shown selected in the Editor. If the error occurred in an INCLUDE file then the INCLUDE statement will be shown selected.

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

5.2 Menu

Pressing the MENU key or touching the Menu icon 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.)

5.2.1 Stop

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

5.2.2 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.

6 A BASIC! Program

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

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

```
PRINT "Hello, World!"
```

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

If you start the BASIC! app, so you are in the Editor, you can type in this one-line program. Then you can select Run from the Editor's menu. BASIC! will run your program. When the program is done running, you see the Console, BASIC!'s, output screen, with Hello, World! printed at the top.

7 Command Description Syntax

7.1 Upper and Lower Case

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

7.2 <nexp>, <sexp> and <lexp>

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

7.3 <nvar>, <svar> and <lvar>

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

7.4 Array[] and Array$[]

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

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

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

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

7.6 {something}

Indicates something optional.

7.7 { A | B |C }

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

```
Text.open {r|w|a}, fn…
```

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

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

7.8 X, …

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

7.9 {,n} …

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

7.10 <statement>

Indicates an executable BASIC! statement. A <statement> is usually a line of code but may occur within other commands such as: **IF** <lexp> **THEN** <statement>.

7.11 Optional Parameters

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

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

8 Numbers

You can type decimal numbers in a BASIC! program:

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

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

Numbers in BASIC! are double-precision floating point (64-bit IEEE 754). This means:

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

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

You can use string functions to convert numbers to strings. STR$(), INT$(), HEX$() and a few others do simple conversions. FORMAT$() and USING$() can do more complex formatting.

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

9 Strings

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

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

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

prints: His name is "Jimbo" Jim Giudice.

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

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

prints:

```
Jim
Giudice
```

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

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

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

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

10 Variables

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

10.1 Variable Names

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

A variable name may be as long as needed.

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

You should avoid using variable names that start with BASIC! command keywords. Such variables are valid under most conditions, as will be explained later in this manual, but their use may cause confusion or errors. For example, **Donut = 5** is interpreted as **Do Nut=5**. BASIC! thus expects this **Do** statement to be followed by an **Until** statement somewhere before the program ends. A list of BASIC! commands can be found in Appendix A. See also the **Let** command and section 13.4 **Parentheses**.

BASIC! statement labels and the names of user-defined functions, both described in the ***Basic Language Reference*** manual, follow the same naming rules as BASIC! variables.

10.2 Variable Types

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

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

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

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

10.3 Scalar and Array Variables

There are two classes of variables: Scalars and Arrays.

10.4 Scalars

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

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

10.5 Arrays

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

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

Arrays can have more than one index or dimension. An array with two dimensions can be thought of as a list of lists. Let's assume that we wanted to assign a list of three traits to every animal in the list of

animals. Such a list for a "Cat" might be "Purrs", "Has four legs" and "Has Tail". We could set up the Traits array to have two dimensions such that Traits$[5,2] = "Has four legs". If someone asked what are the traits of cat, search Animals$[index] until "Cat" is found at index=5. Index=5 can then be used to access Traits[index,[{1|2|3}].

BASIC! arrays can have any number of dimensions of any size.

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

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

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

10.6 Array Segments

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

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

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

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

11 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 implementation of Lists, Bundles, Stacks and Queues.

11.1 What is a Pointer

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

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

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

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

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

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

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

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

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

in **SoundPool**, and other things described in the ***Basic Language Reference***.

11.2 Lists

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

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

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

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

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

11.3 Bundles

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

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

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

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

11.3.1 Bundle Auto-Create

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

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

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

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

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

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

11.4 Stacks

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

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

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

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

11.5 Queues

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

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

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

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

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

12 Comments

12.1 ! - 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.

12.2 Rem - Single Line Comment (legacy)

If the first three characters in a line are "Rem", "REM", or even "rEm", BASIC! considers the entire line a comment and ignores it. If "Rem" appears elsewhere in the line it does not indicate a comment.

12.3 !! - Block Comment

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

12.4 % - Middle of Line Comment

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

13 Expressions

13.1 Numeric Expression <nexp>

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

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

The next few sections define all of the terms.

13.1.1 Numeric Operators <noperator>

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

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

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

13.1.2 Numeric Expression Examples

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

13.1.3 Pre- and Post-Increment Operators

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

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

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

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

13.2 String Expression <sexp>

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

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

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

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

13.3 Logical Expression <lexp>

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

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

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

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

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

13.3.1 Logical Operators

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

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

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

13.3.2 Examples of Logical Expressions

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

13.4 Parentheses

Parentheses can be used to override operator precedence.

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

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

14 Assignment Operations

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

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

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

14.1 Let

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

```
LET <nvar> = <nexp>
```

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

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

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

is seen by BASIC! as

```
LET ter$ = "B"
```

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

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

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

```
IF 1 < 2 THEN LET letter$ = "B"
```

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

14.2 OpEqual Assignment Operations

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

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

Here are some examples:

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

15 Command List

! - Single Line Comment

!! - Block Comment

- Format Line

% - Middle of Line Comment

? {<exp> {,|;}} ...

ABS(<nexp>)

ACOS(<nexp>)

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

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

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

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

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

Array.delete Array[], Array$[] ...

Array.dims Source[]{, {Dims[]}{, NumDims}}

Array.fill Array$[{<start>,<length>}], <sexp>

Array.fill Array[{<start>,<length>}], <nexp>

Array.length n, Array$[{<start>,<length>}]

Array.length n, Array[{<start>,<length>}]

Array.load Array$[], <sexp>, ...

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

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

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

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

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

Array.search Array$[{<start>,<length>}], <value_sexp>, <result_nvar>{,<start_nexp>}

Array.search Array[{<start>,<length>}], <value_nexp>, <result_nvar>{,<start_nexp>}

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

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

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

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

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

ASCII(<sexp>{, <index_nexp>})

ASIN(<nexp>)

ATAN(<nexp>)

ATAN2(<nexp_y>, <nexp_x>)

Audio.isdone <lvar>

Audio.length <length_nvar>, <aft_nexp>

Audio.load <aft_nvar>, <filename_sexp>

Audio.loop

Audio.pause

Audio.play <aft_nexp>

Audio.position.current <nvar>

Audio.position.seek <nexp>

Audio.record.start <fn_svar>

Audio.record.stop

Audio.release <aft_nexp>

Audio.stop

Audio.volume <left_nexp>, <right_nexp>

Back.resume

BACKGROUND()

Background.resume

BAND(<nexp1>, <nexp2>)

BIN$(<nexp>)

BIN(<sexp>)

BNOT(<nexp>)

BOR(<nexp1>, <nexp2>)

Browse <url_sexp>

Bt.close

Bt.connect {0|1}

Bt.device.name <svar>

Bt.disconnect

Bt.onReadReady.resume

Bt.open {0|1}

Bt.read.bytes <svar>

Bt.read.ready <nvar>

Bt.reconnect

Bt.set.UUID <sexp>

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

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

Bundle.clear <pointer_nexp>

Bundle.contain <pointer_nexp>, <key_sexp> , <contains_nvar>

Bundle.create <pointer_nvar>

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

Bundle.keys <bundle_ptr_nexp>, <list_ptr_nexp>

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

Bundle.remove <pointer_nexp>, <key_sexp>

Bundle.type <pointer_nexp>, <key_sexp>, <type_svar>

BXOR(<nexp1>, <nexp2>)

Byte.close <file_table_nexp>

Byte.copy <file_table_nexp>,<output_file_sexp>

Byte.eof <file_table_nexp>, <lvar>

Byte.open {r|w|a}, <file_table_nvar>, <path_sexp>

Byte.position.get <file_table_nexp>, <position_nexp>

Byte.position.mark {{<file_table_nexp>}{, <marklimit_nexp>}}

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

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

Byte.read.byte <file_table_nexp> {,<nvar>}...

Byte.read.number <file_table_nexp> {,<nvar>...}

Byte.truncate <file_table_nexp>,<length_nexp>

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

Byte.write.byte <file_table_nexp> {{,<nexp>}...{,<sexp>}}

Byte.write.number <file_table_nexp> {,<nexp>}...

Call <user_defined_function>

CBRT(<nexp>)

CEIL(<nexp>)

CHR$(<nexp>, ...)

Clipboard.get <svar>

Clipboard.put <sexp>

CLOCK()

Cls

Console.front

Console.line.count <count_nvar >

Console.line.text <line_nexp>, <text_svar>

Console.line.touched <line_nvar> {, <press_lvar>}

Console.save <filename_sexp>

Console.title { <title_sexp>}

ConsoleTouch.resume

COS(<nexp>)

COSH(<nexp>)

D_U.break

D_U.continue

Debug.dump.array Array[]

Debug.dump.bundle <bundlePtr_nexp>

Debug.dump.list <listPtr_nexp>

Debug.dump.scalars

Debug.dump.stack <stackPtr_nexp>

Debug.echo.off

Debug.echo.on

Debug.off

Debug.on

Debug.print

Debug.show

Debug.show.array Array[]

Debug.show.bundle <bundlePtr_nexp>

Debug.show.list <listPtr_nexp>

Debug.show.program

Debug.show.scalars

Debug.show.stack <stackPtr_nexp>

Debug.show.watch

Debug.watch var, ...

DECODE$(<charset_sexp>, <buffer_sexp>)

DECODE$(<type_sexp>, {<qualifier_sexp>}, <source_sexp>)

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

Device <nexp>|<nvar>

Device <svar>

Device.Language <svar>

Device.Locale <svar>

Dialog.message {<title_sexp>}, {<message_sexp>}, <sel_nvar> {, <button1_sexp>{, <button2_sexp>{, <button3_sexp>}}}

Dialog.select <sel_nvar>, < Array$[]>|<list_nexp>, {,<title_sexp>}

Dim Array [n, n, ...], Array$[n, n, ...] ...

Do / Until <lexp>

Echo.off

Echo.on

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

ENCODE$(<charset_sexp>, <source_sexp>)

ENCODE$(<type_sexp>, {<qualifier_sexp>}, <source_sexp>)

Encrypt {<pw_sexp>}, <source_sexp>, <encrypted_svar>

End{ <msg_sexp>}

ENDS_WITH(<sub_sexp>, <base_sexp>)

Exit

EXP(<nexp>)

F_N.break

F_N.continue

File.delete <lvar>, <path_sexp>

File.dir <path_sexp>, Array$[] {, <dirmark_sexp>}

File.exists <lvar>, <path_sexp>

File.mkdir <path_sexp>

File.rename <old_path_sexp>, <new_path_sexp>

File.root <svar>

File.size <size_nvar>, <path_sexp>

File.type <type_svar>, <path_sexp>

FLOOR(<nexp>)

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

Fn.end

Fn.rtn <sexp>|<nexp>

Font.clear

Font.delete {<font_ptr_nexp>}

Font.load <font_ptr_nvar>, <filename_sexp>

For <nvar> = <nexp> To <nexp> {Step <nexp>} / Next

FORMAT$(<pattern_sexp>, <nexp>)

FORMAT_USING$(<locale_sexp>, <format_sexp> { , <exp>}...)

FRAC(<nexp>)

Ftp.cd <new_directory_sexp>

Ftp.close

Ftp.delete <filename_sexp>

Ftp.dir <list_nvar>

Ftp.get <source_sexp>, <destination_sexp>

Ftp.mkdir <directory_sexp>

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

Ftp.put <source_sexp>, <destination_sexp>

Ftp.rename <old_filename_sexp>, <new_filename_sexp>

Ftp.rmdir <directory_sexp>

GETERROR$()

GoSub <index_nexp>, <label>... / Return

GoSub <label> / Return

GoTo <index_nexp>, <label>...

GoTo <label>

Gps.accuracy <nvar>

Gps.altitude <nvar>

Gps.bearing <nvar>

Gps.close

Gps.latitude <nvar>

Gps.location {{<time_nvar>}, {<prov_svar>}, {<count_nvar}, {<acc_nvar>}, {<lat_nvar>}, {<long_nvar>}, {<alt_nvar>}, {<bear_nvar>}, {<speed_nvar>}}

Gps.longitude <nvar>

Gps.open {{<status_nvar>},{<time_nexp>},{<distance_nexp>}}

Gps.provider <svar>

Gps.satellites {{<count_nvar>}, {<sat_list_nexp>}}

Gps.speed <nvar>

Gps.status {{<status_var>}, {<infix_nvar>},{inview_nvar}, {<sat_list_nexp>}}

Gps.time <nvar>

Gr.arc <obj_nvar>, left, top, right, bottom, start_angle, sweep_angle, fill_mode

Gr.bitmap.create <bitmap_ptr_nvar>, width, height

Gr.bitmap.crop <new_bitmap_ptr_nvar>, <source_bitmap_ptr_nexp>, <x_nexp>, <y_nexp>, <width_nexp>, <height_nexp>

Gr.bitmap.delete <bitmap_ptr_nexp>

Gr.bitmap.draw <object_ptr_nvar>, <bitmap_ptr_nexp>, x , y

Gr.bitmap.drawinto.end

Gr.bitmap.drawinto.start <bitmap_ptr_nexp>

Gr.bitmap.fill <bitmap_ptr_nexp>, <x_nexp>, <y_nexp>

Gr.bitmap.load <bitmap_ptr_nvar>, <file_name_sexp>

Gr.bitmap.save <bitmap_ptr_nvar>, <filename_sexp>{, <quality_nexp>}

Gr.bitmap.scale <new_bitmap_ptr_nvar>, <bitmap_ptr_nexp>, width, height {, <smoothing_lexp> }

Gr.bitmap.size <bitmap_ptr_nexp>, width, height

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

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

Gr.brightness <nexp>

Gr.camera.autoshoot <bm_ptr_nvar>{, <flash_ mode_nexp> {, focus_mode_nexp} }

Gr.camera.manualShoot <bm_ptr_nvar>{, <flash_ mode_nexp> {, focus_mode_nexp} }

Gr.camera.select 1|2

Gr.camera.shoot <bm_ptr_nvar>

Gr.circle <obj_nvar>, x, y, radius

Gr.clip <object__ptr_nexp>, <left_nexp>,<top_nexp>, <right_nexp>, <bottom_nexp>{, <RO_nexp>}

Gr.close

Gr.cls

Gr.color {{alpha}{, red}{, green}{, blue}{, style}{, paint}}

Gr.front flag

Gr.get.bmpixel <bitmap_ptr_nvar>, x, y, alpha, red, green, blue

Gr.get.params <object_ptr_nexp>, <param_array$[]>

Gr.get.pixel x, y, alpha, red, green, blue

Gr.get.position <object_ptr_nexp>, x, y

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

Gr.get.type <object_ptr_nexp>, <type_svar>

Gr.get.value <object_ptr_nexp> {, <tag_sexp>, <value_nvar | value_svar>}...

Gr.getDL <dl_array[]> {, <keep_all_objects_lexp> }

Gr.group <object_number_nvar>{, <obj_nexp>}...

Gr.group.getDL <object_number_nvar>

Gr.group.list <object_number_nvar>, <list_ptr_nexp>

Gr.group.newDL <object_number_nvar>

Gr.hide <object_number_nexp>

Gr.line <obj_nvar>, x1, y1, x2, y2

Gr.modify <object_ptr_nexp> {, <tag_sexp>, <value_nexp | value_sexp>}...

Gr.move <object_ptr_nexp>{{, dx}{, dy}}

Gr.newDL <dl_array[{<start>,<length>}]>

Gr.onGrTouch.resume

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

Gr.orientation <nexp>

Gr.oval <obj_nvar>, left, top, right, bottom

Gr.paint.copy {{<src_nexp>}{, <dst_nexp>}}

Gr.paint.get <object_ptr_nvar>

Gr.paint.reset {<nexp>}

Gr.point <obj_nvar>, x, y

Gr.poly <obj_nvar>, list_pointer {,x,y}

Gr.rect <obj_nvar>, left, top, right, bottom

Gr.render

Gr.rotate.end {<obj_nvar>}

Gr.rotate.start angle, x, y{,<obj_nvar>}

Gr.save <filename_sexp> {,<quality_nexp>}

Gr.scale x_factor, y_factor

Gr.screen width, height{, density }

Gr.screen.to_bitmap <bm_ptr_nvar>

Gr.set.antialias {{<lexp>}{,<paint_nexp>}}

Gr.set.pixels <obj_nvar>, pixels[{<start>,<length>}] {,x,y}

Gr.set.stroke {{<nexp>}{,<paint_nexp>}}

Gr.show <object_number_nexp>

Gr.show.toggle <object_number_nexp>

Gr.statusbar {<height_nvar>} {, showing_lvar}

Gr.statusbar.show <nexp>

Gr.text.align {{<type_nexp>}{,<paint_nexp>}}

Gr.text.bold {{<lexp>}{,<paint_nexp>}}

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

Gr.text.height {<height_nvar>} {, <up_nvar>} {, <down_nvar>}

Gr.text.setfont {{<font_ptr_nexp>|<font_family_sexp>} {, <style_sexp>} {,<paint_nexp>}}

Gr.text.size {{<size_nexp>}{,<paint_nexp>}}

Gr.text.skew {{<skew_nexp>}{,<paint_nexp>}}

Gr.text.strike {{<lexp>}{,<paint_nexp>}}

Gr.text.typeface {{<nexp>} {, <style_nexp>} {,<paint_nexp>}}

Gr.text.underline {{<lexp>}{,<paint_nexp>}}

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

Gr.touch touched, x, y

Gr.touch2 touched, x, y

GR_COLLISION(<object_1_nvar>, <object_2_nvar>)

GrabFile <result_svar>, <path_sexp>{, <unicode_flag_lexp>}

GrabURL <result_svar>, <url_sexp>{, <timeout_nexp>}

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

HEX$(<nexp>)

HEX(<sexp>)

Home

Html.clear.cache

Html.clear.history

Html.close

Html.get.datalink <data_svar>

Html.go.back

Html.go.forward

Html.load.string <html_sexp>

Html.load.url <file_sexp>

Html.open {<ShowStatusBar_lexp> {, <Orientation_nexp>}}

Html.orientation <nexp>

Html.post <url_sexp>, <list_nexp>

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

HYPOT(<nexp_x>, <nexp_y)

If / Then / Else

If / Then / Else / Elseif / Endif

Include FilePath

Inkey$ <svar>

Input {<prompt_sexp>}, <result_var>{, {<default_exp>}{,<canceled_nvar>}}

INT$(<nexp>)

INT(<nexp>)

IS_IN(<sub_sexp>, <base_sexp>{, <start_nexp>}

IS_NUMBER(<sexp>)

Join <source_array$[]>, <result_svar> {, <separator_sexp>{, <wrapper_sexp}}

Join.all <source_array$[]>, <result_svar> {, <separator_sexp>{, <wrapper_sexp}}

Kb.hide

Kb.resume

Kb.show

Kb.showing <lvar>

Kb.toggle

Key.resume

LEFT$(<sexp>, <count_nexp>)

LEN(<sexp>)

Let

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

List.add.array numeric_list_pointer, Array[{<start>,<length>}]

List.add.array string_list_pointer, Array$[{<start>,<length>}]

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

List.clear <pointer_nexp>

List.create N|S, <pointer_nvar>

List.get <pointer_nexp>, <index_nexp>, <var>

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

List.remove <pointer_nexp>,<index_nexp>

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

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

List.size <pointer_nexp>, <nvar>

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

List.type <pointer_nexp>, <svar>

LOG(<nexp>)

LOG10(<nexp>)

LOWER$(<sexp>)

LowMemory.resume

MAX(<nexp>, <nexp>)

MenuKey.resume

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

MIN(<nexp>, <nexp>)

mkdir <path_sexp>

MOD(<nexp1>, <nexp2>)

MyPhoneNumber <svar>

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

OCT$(<nexp>)

OCT(<sexp>)

OnBackground:

OnBackKey:

OnBtReadReady:

OnConsoleTouch:

OnError:

OnGrTouch:

OnKbChange:

OnKeyPress:

OnLowMemory:

OnMenuKey:

OnTimer:

Pause <ticks_nexp>

Phone.call <sexp>

Phone.dial <sexp>

Phone.info <nexp>|<nvar>

Phone.rcv.init

Phone.rcv.next <state_nvar>, <number_svar>

PI()

Popup <message_sexp>{{, <x_nexp>}{, <y_nexp>}{, <duration_lexp>}}

POW(<nexp1>, <nexp2>)

Print {<exp> {,|;}} ...

Program.info <nexp>|<nvar>

RANDOMIZE({<nexp>})

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

Read.from <nexp>

Read.next <var>, ...

Rem

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

RIGHT$(<sexp>, <count_nexp>)

Ringer.get.mode <nvar>

Ringer.get.volume <vol_nvar> { , <max_nvar>}

Ringer.set.mode <nexp>

Ringer.set.volume <nexp>

RND()

ROUND(<value_nexp>{, <count_nexp>{, <mode_sexp>}})

Run <filename_sexp>{, <data_sexp>}

Screen rotation, size[], realsize[], density

Screen.rotation <nvar>

Screen.size size[], realsize[], density

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

Sensors.close

Sensors.list <sensor_array$[]>

Sensors.open <type_nexp>{:<delay_nexp>}{, <type_nexp>{:<delay_nexp>}, ...}

Sensors.read sensor_type, p1, p2, p3

SGN(<nexp>)

SHIFT(<value_nexp>, <bits_nexp>)

SIN(<nexp>)

SINH(<nexp>)

Sms.rcv.init

Sms.rcv.next <svar>

Sms.send <number_sexp>, <message_sexp>

Socket.client.close

Socket.client.connect <server_sexp>, <port_nexp> { , <wait_lexp> }

Socket.client.read.file <file_nexp>

Socket.client.read.line <line_svar>

Socket.client.read.ready <nvar>

Socket.client.server.ip <svar>

Socket.client.status <status_nvar>

Socket.client.write.bytes <sexp>

Socket.client.write.file <file_nexp>

Socket.client.write.line <line_sexp>

Socket.myIP <array$[]>{, <nvar>}

Socket.myIP <svar>

Socket.server.client.ip <nvar>

Socket.server.close

Socket.server.connect {<wait_lexp>}

Socket.server.create <port_nexp>

Socket.server.disconnect

Socket.server.read.file <file_nexp>

Socket.server.read.line <svar>

Socket.server.read.ready <nvar>

Socket.server.status <status_nvar>

Socket.server.write.bytes <sexp>

Socket.server.write.file <file_nexp>

Socket.server.write.line <line_sexp>

Soundpool.load <soundID_nvar>, <file_path_sexp>

Soundpool.open <MaxStreams_nexp>

Soundpool.pause <streamID_nexp>

Soundpool.play <streamID_nvar>, <soundID_nexp>, <rightVolume_nexp>, <leftVolume_nexp>, <priority_nexp>, <loop_nexp>, <rate_nexp>

Soundpool.release

Soundpool.resume <streamID_nexp>

Soundpool.setpriority <streamID_nexp>, <priority_nexp>

Soundpool.setrate <streamID_nexp>, <rate_nexp>

Soundpool.setvolume <streamID_nexp>, <leftVolume_nexp>, <rightVolume_nexp>

Soundpool.stop <streamID_nexp>

Soundpool.unload <soundID_nexp>

Split <result_array$[]>, <sexp> {, <test_sexp>}

Split.all <result_array$[]>, <sexp> {, <test_sexp>}

Sql.close <DB_pointer_nvar>

Sql.delete <DB_pointer_nvar>, <table_name_sexp>{,<where_sexp>{,<count_nvar>} }

Sql.drop_table <DB_pointer_nvar>, <table_name_sexp>

Sql.exec <DB_pointer_nvar>, <command_sexp>

Sql.insert <DB_pointer_nvar>, <table_name_sexp>, C1$, V1$, C2$, V2$, ...,CN$, VN$

Sql.new_table <DB_pointer_nvar>, <table_name_sexp>, C1$, C2$, ...,CN$

Sql.next doneFlag, cursorVar {{, svar}...{, array$[]{, nColVar}}}

Sql.open <DB_pointer_nvar>, <DB_name_sexp>

Sql.query <cursor_nvar>, <DB_pointer_nvar>, <table_name_sexp>, <columns_sexp> {, <where_sexp> {,<order_sexp>} }

Sql.query.length <length_nvar>, <cursor_nvar>

Sql.query.position <position_nvar>, <cursor_nvar>

Sql.raw_query <cursor_nvar>, <DB_pointer_nvar>, <query_sexp>

Sql.update <DB_ptr_nvar>, <table_name_sexp>, C1$, V1$, C2$, V2$,...,CN$, VN${: <where_sexp>}

SQR(<nexp>)

Stack.clear <ptr_nexp>

Stack.create N|S, <ptr_nvar>

Stack.isEmpty <ptr_nexp>, <nvar>

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

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

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

Stack.type <ptr_nexp>, <svar>

STARTS_WITH(<sub_sexp>, <base_sexp>{, <start_nexp>}

STR$(<nexp>)

STT.listen

STT.results <string_list_ptr_nexp>

Su.close

Su.open

Su.read.line <svar>

Su.read.ready <nvar>

Su.write <sexp>

Sw.begin <exp>

Sw.break

Sw.case <exp >, ...

Sw.case <op><exp >

Sw.default

Sw.end

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

System.close

System.open

System.read.line <svar>

System.read.ready <nvar>

System.write <sexp>

TAN(<nexp>)

Text.close <file_table_nexp>

Text.eof <file_table_nexp>, <lvar>

Text.input <svar>{, { <text_sexp>} , <title_sexp> }

Text.open {r|w|a}, <file_table_nvar>, <path_sexp>

Text.position.get <file_table_nexp>, <position_nvar>

Text.position.mark {{<file_table_nexp>}{, <marklimit_nexp>}}

Text.position.set <file_table_nexp>, <position_nexp>

Text.readln <file_table_nexp> {,<svar>}...

Text.writeln <file_table_nexp>, {<exp> {,|;}} ...

TGet <result_svar>, <prompt_sexp> {, <title_sexp>}

Time {<time_nexp>,} Year$, Month$, Day$, Hour$, Minute$, Second$, WeekDay, isDST

TIME()

TIME(<year_exp>, <month_exp>, <day_exp>, <hour_exp>, <minute_exp>, <second_exp>)

Timer.clear

Timer.resume

Timer.set <interval_nexp>

TimeZone.get <tz_svar>

TimeZone.list <tz_list_pointer_nexp>

TimeZone.set { <tz_sexp> }

TODEGREES(<nexp>)

Tone <frequency_nexp>, <duration_nexp>{, <duration_chk_lexp}

TORADIANS(<nexp>)

TRIM$(<sexp>{, <test_sexp>})

TTS.init

TTS.speak <sexp> {, <wait_lexp>}

TTS.speak.toFile <sexp> {, <path_sexp>}

TTS.stop

UCODE(<sexp>{, <index_nexp>})

UnDim Array[], Array$[], ...

UPPER$(<sexp>)

USING$({<locale_sexp>} , <format_sexp> { , <exp>}...)

VAL(<sexp>)

VERSION$()

Vibrate <pattern_array[{<start>,<length>}]>,<nexp>

VolKeys.off

VolKeys.on

W_R.break

W_R.continue

WakeLock <code_nexp>{, <flags_nexp>

While <lexp> / Repeat

WiFi.info {{<SSID_svar>}{, <BSSID_svar>}{, <MAC_svar>}{, <IP_var>}{, <speed_nvar>}}

WifiLock <code_nexp>

WORD$(<source_sexp>, <n_nexp> {, <test_sexp>})

Zip.close, <file_table_nexp>

Zip.count <path_sexp>, <nvar}

Zip.dir <path_sexp>, Array$[] {,<dirmark_sexp>}

Zip.open {r|w|a}, <file_table_nvar>, <path_sexp>

Zip.read <file_table_nexp> ,<buffer_svar>, <file_name_sexp>

16 Sample Programs

The programs are loaded into "<pref base drive>/rfo-basic/source/Sample_Programs" when a new release of BASIC! is installed. You can access them by selecting **Menu→Load**. Tap the "Sample_Programs" line. The sample programs will be listed and can be loaded.

If you load and save one of these programs, the program will be saved in "<pref base drive>/rfo-basic/source/" not in "<pref base drive>/rfo-basic/source/Sample_Programs".

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

- Select Menu→Delete
- Navigate to "rfo-basic/source/Sample_Programs/"
- Delete the "f01_vxx.xx_read_me file"
- Exit BASIC! using **Menu→Exit** or **Menu→More→Exit.**

17 Launcher Shortcut Tutorial

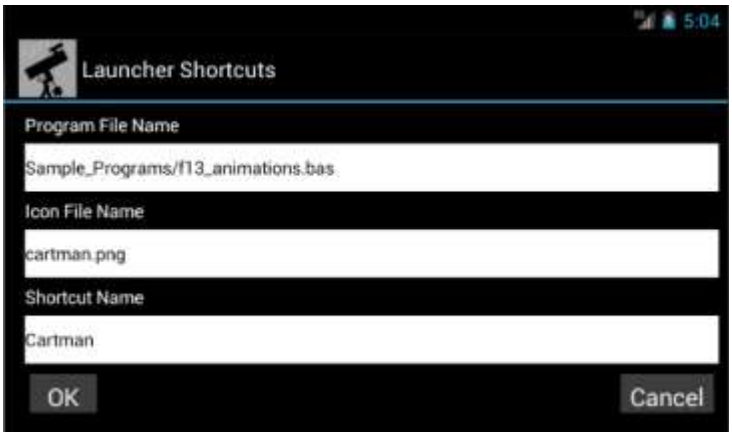
17.1 Introduction

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. See chapter 18 Basic CompilerBasic Compiler.

17.2 How to Make a Shortcut Application

1. Select the Apps Page.
2. At the top of the screen, tap Widgets.
3. Scroll horizontally until you see the BASIC! icon that says Launcher Shortcuts.
4. Touch and hold that entry. It will be moved to the Home page.
5. This screen will appear:



6. Fill out the Form exactly as shown.
   - The Program File Name is Sample_Programs/f13_animations.bas.
   - The Icon File Name is cartman.png.
   - The Shortcut Name is Cartman.
7. Tap OK.
8. You should see something like this on your HOME screen:

9. Tap the Cartman Shortcut.

10. BASIC! will start and run the Cartman Jumping Demo.

17.3 What you need to know

- The icon image file must be located in the "<pref base drive>/rfo-basic/data/" directory.
- The program that you are going to run must be in the "source" directory or one of its sub directories. In this example, the file was located in the Sample_Programs(d) subdirectory of the "source(d)" directory.
- The icon should be a .png file. A Google search for "icon" will reveal thousands for free icons. Just copy your icon into "rfo-basic/data" on the SD card.
- Be very careful to correctly spell the names of the program and icon files. BASIC! does not check to see if these files actually exist during the "compile" process. If you enter the name of an icon file that does not exist, your shortcut will have the generic Android icon. If the file name you specified does not exist, when you tap the Shortcut you will see an error message in the form of program file in the Editor.
- The Shortcut name should be nine (9) characters or less. Android will not show more than nine characters.
- You can create as many shortcuts as you home screen(s) can handle.
- Tapping "Cancel" in the Launcher Shortcuts dialog will simply cancel the operation and return to the home screen.
- If you plan to use a BASIC! Launcher Shortcut, you should always exit BASIC! using **Exit→Menu** or **Menu→More→Exit**. If a Launched program is running, tapping BACK once or twice will exit BASIC back to the Home Screen.

18 Basic Compiler

19 BASIC! Distribution License

19.1 GNU General Public 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

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

19.2 Apache Commons

Portions of BASIC! use Apache Commons.

Apache Commons Net

Copyright 2001-2012 The Apache Software Foundation

This product includes software developed by

The Apache Software Foundation (http://www.apache.org/).