

### Boxplots (compare) ▾

19 —



19 —



## Results

### Robust

P90	7.6
Q3	7
Median	7
Q1	6
P10	5.4

### Classical

$\bar{x}$	(mean)	6.6
N	(count)	5
$\Sigma$	(sum)	33
$s$	(sample stddev)	6.3087
$\sigma_n$	(pop. stddev)	5.6427
RSD	(rel. stddev)	0.9559

# Guide to using Best Calculator

for windows and windows Phone

Version 3.20, December 2021

BC BASIC Reference manual and tutorial  
including

The Best Calculator Reference Manual

■ Version 3.20, December 2021 ■

© 2016, 2017, 2018 Peter D Smith

The Best Calculator app is available for Windows and Windows Phone  
from the Microsoft App Store at

<https://www.microsoft.com/en-us/store/apps/best-calculator/9wzdnrcrdfd6x>

Visit the Best Calculator web site at  
<https://bestcalculator.wordpress.com/>

## TABLE OF CONTENTS

---

1	A Quick Tour of Best Calculator .....	23
2	Numbers and Common Calculations .....	25
2.1	Simple Arithmetic.....	25
2.2	Chain calculations.....	25
2.3	Algebraic Entry and Parentheses.....	25
2.4	Editing errors and clearing the display .....	26
3	Memory Keys .....	27
3.1	Memory operation .....	27
3.2	Example .....	27
3.3	Memory Add and Subtract .....	28
4	More Math .....	29
4.1	Square ( $x^2$ ) key.....	29
4.2	Square Root ( $\sqrt{x}$ ) key .....	29
4.3	Inverse ( $1/x$ ) key .....	29
4.4	Change Sign ( $\pm$ ) key .....	29
5	Scientific Notation.....	30
5.1	Using the EE key .....	30
6	Percent Key % .....	31
6.1	Percent (%) key.....	31
6.2	Calculating sales tax .....	31
6.3	Calculating sales with a percent discount .....	31
7	Trigonometry keys on the Advanced calculator .....	32
7.1	Calculate in degrees or radians .....	32
7.2	Sin, Cos, Tan .....	32
7.3	Inverse Sin, Cos, Tan .....	33
8	Logarithms on the Advanced calculator .....	34
9	Powers, roots, $x!$ , mod on the Advanced calculator .....	35
9.1	Factorial ( $x!$ ) key .....	35

	Page   4
<b>Guide to Using Best Calculator</b>	
9.2 Mod Key .....	35
9.3 Cube ( $x^3$ ) and Cube Root ( $\sqrt[3]{x}$ ) keys.....	35
9.4 Arbitrary Power ( $x^y$ ) and Root ( $y\sqrt{x}$ ) keys.....	36
10 Rounding and abs on the Advanced calculator .....	37
11 Random Numbers on the Advanced calculator .....	38
12 Formatting.....	39
12.1 Standard formatting .....	39
12.2 Exponent formatting .....	40
12.3 Fixed formatting .....	40
12.4 Natural formatting.....	40
12.5 Percent formatting .....	41
13 Constants .....	42
13.1 $\pi$ and e.....	42
13.2 $g_n$ c and $N_a$ .....	42
13.3 NaN, $\infty$ and $-\infty$ .....	42
14 Memory Page .....	43
14.1 Show the operation of Memory0.....	43
14.2 Saving and Roaming .....	43
14.3 Name a memory cell .....	43
14.4 Get from display and Copy to display.....	44
14.5 Clear .....	44
14.6 Memory and BC Basic.....	44
15 Dates Page.....	45
15.1 Compare Dates .....	45
15.2 Convert to Gregorian.....	45
15.3 Add days .....	46
15.4 Subtract days.....	46
16 Hex, Decimal, Octal, Binary on the Programmer's calculator.....	47
16.1 Setting the mode .....	47
16.2 Converting between bases .....	47

	Page   5
Guide to Using Best Calculator	
17 Bit Operators on the Programmer's calculator.....	48
17.1 B# (count bits) .....	48
17.2 Inverse (~) and 2's complement.....	48
17.3 And (&), Or ( ) , Xor (^).....	48
18 Bytes and Swabbing on the Programmer's calculator .....	49
18.1 Bytes .....	49
18.2  SWAB (Swap Bytes) key.....	49
19 Shift operators on the Programmer's calculator .....	50
20 Programmer's Math.....	51
21 Statistical calculator .....	52
21.1 Parts of the Statistical calculator screen .....	52
21.2 Entering Data.....	53
21.3 Classical Statistics (Results) .....	54
Robust Statistics (Results) .....	55
Regression (Results) .....	56
21.4 Compare with T-Tests (Results).....	57
21.5 Boxplots (Graph).....	58
21.6 XY Scatter-plots (Graph) .....	59
22 Conversions.....	60
22.1 Converting .....	60
22.2 Copy to and from the Calculator results display .....	60
22.3 Area conversions .....	61
22.4 Energy Conversions .....	61
22.5 Length Conversions .....	61
22.6 Temperature Conversions .....	62
22.7 Weight .....	62
22.8 Farm Volumes (US).....	63
23 Ascii Table .....	64
24 Unicode .....	65
24.1 Search Rules .....	65

	Page   6
Guide to Using Best Calculator	
24.2     Copying characters .....	65
25       Advanced Windows Features.....	66
25.1    Shortcut on the desktop.....	66
25.2    Set as the Calculator Key .....	67
26       Appearance and Alignment .....	68
26.1    About the Antique fonts.....	69
27       A brief historical note about BASIC.....	70
28       Equation Input: your first program .....	71
28.1    What is a program? .....	71
28.2    How do I run my program? .....	71
28.3    How can I print some text? .....	71
28.4    How can my program use the calculator value? .....	72
28.5    How can I write an equation? .....	72
28.6    How can I write to the calculator? .....	72
28.7    Why do some examples start with an = sign? .....	73
28.8    How can I make several different programs?.....	73
28.9    What are all the keys on the screen? .....	73
28.10   How can I learn more? .....	74
29       Sigma Function: advanced programming .....	75
30       What all can you do in the BC BASIC environment?.....	77
30.1    All the main edit dialogs .....	77
30.2    Library of Packages.....	78
30.3    Library Properties (Import BC BASIC package) .....	80
30.4    Bind a program to a key .....	81
30.5    About this package .....	82
30.6    List of programs.....	84
30.7    About this program .....	86
30.8    Edit Program.....	88
30.9    Output screen.....	90
30.10   Running ✓ Tests in BC BASIC.....	91

Guide to Using Best Calculator	Page   7
31 BASIC Language Reference .....	93
31.1 Program structure .....	93
31.2 Flags.....	94
31.3 Numbers and Strings and Variables .....	94
31.3.1 Numbers.....	94
31.3.2 Strings.....	95
31.3.3 Variables, GLOBAL, and DIM .....	96
31.4 <expression> overview .....	98
31.4.1 Quick introduction to expressions .....	98
31.4.2 Expression Rules.....	98
31.4.3 negate, power, root prefix and postfix operators .....	100
31.4.4 Operators + - * / and more .....	100
31.4.5 The INKEY\$ expression.....	104
31.4.6 The INPUT expression .....	105
31.5 Math Functions.....	105
31.5.1 Trigonometry functions SIN COS TAN ASN ACS ATN .....	105
31.5.2 Logarithm and Power functions LN EXP SQR .....	106
31.5.3 Rounding and sign functions SGN ABS INT .....	106
31.6 String functions (LEFT, MID, RIGHT, LEN, CHR\$, CODE, SPC, VAL) .....	107
31.6.1 LEFT (string, count)., MID (string, index, count) and RIGHT (string, count)	107
31.6.2 LEN string .....	108
31.6.3 CHR, CODE (and ASC) .....	108
31.6.4 SPC (length).....	110
31.6.5 VAL(string).....	110
31.7 <statement> overview.....	110
31.8 Packages and Programs.....	112
31.8.1 Inside a package file .....	113

	Page   8
Guide to Using Best Calculator	
31.9    Picking a package for your program .....	113
31.9.1    Creating common functions for several programs .....	113
31.9.2    Exporting packages .....	113
31.9.3    Pretty-print a package.....	114
31.10    All of the Special Symbols .....	115
32    BASIC Statements Reference .....	116
32.1    BEEP [duration, pitch] .....	116
32.2    [CALL] <function> (<expression>, ...)	117
32.3    CLS [<color>,<color>] and PAPER <color> .....	117
32.4    CONSOLE <expression> [, <expression>].....	119
32.5    DATA – see the READ and DATA section .....	120
32.6    DIM <name> ([size [, size]]) and array methods.....	120
32.7    DIM'd array methods and properties.....	122
32.7.1    Add(data1, data2, ...).....	122
32.7.2    AddRow(data1, data2, ...) .....	122
32.7.3    Clear() method .....	123
32.7.4    Count property.....	123
32.7.5    Fill (value).....	123
32.7.6    HasKey(name) method .....	123
32.7.7    Max and Min properties .....	123
32.7.8    MaxOf(column) and MinOf(column) methods .....	123
32.7.9    Mean method .....	123
32.7.10    SetProperty method.....	123
32.7.11    SumOfSquares method .....	124
32.7.12    Add() method and MaxCount and RemoveAlgorithm properties 124	
32.7.13    AddRow() method.....	125
32.8    DUMP .....	125
32.9    FOR <variable> = <start> TO <end> [STEP <step>] ... NEXT <variable> .....	126

Guide to Using Best Calculator		Page   9
32.10	FOREVER [WAIT STOP] .....	129
32.11	FUNCTION <name> ( <args> ) ... RETURN [<value>].....	129
32.12	GLOBAL <variable>.....	130
32.13	GOSUB <linenumber> and RETURN .....	130
32.14	GOTO <linenumber> .....	131
32.15	IF (<expression>) THEN <statement> [ELSE <statement>].....	131
32.16	IF (<expression>) ... [ELSE ...] ENDIF .....	132
32.17	IMPORT FUNCTIONS FROM “program” .....	132
32.18	INPUT <variable> [,<variable>...]	133
32.19	(LET) <variable> = <expression>.....	134
32.20	PAUSE <frames>.....	135
32.21	PLAY <music> .....	135
32.21.1	Simple scales .....	136
32.21.2	Switching Instrument with “I” .....	136
32.21.3	PLAY ONNOTE “function” .....	136
32.21.4	PLAY STOP .....	136
32.21.5	PLAY WAIT .....	136
32.22	PRINT [AT row,col] <expression> [ (, or ;) [AT row,col] <expression>]* ....	137
32.23	RAND <seed> & the RND value .....	139
32.24	READ, RESTORE and DATA .....	140
32.25	REM comment words to the end of the line .....	141
32.26	SPEAK [VOICE <voice>] <text> .....	141
32.27	STOP [SILENT][value] and END .....	142
33	Extensions Reference .....	143
33.1	Calculator.Value and Calculator.Message Extension .....	143
33.2	Data Extension .....	144
33.2.1	Data location values .....	144
33.2.2	Data.GetLocations (“name”) .....	144
33.2.3	Data.Picklocation() .....	145
33.3	DateTime Extension .....	146

Guide to Using Best Calculator	Page   10
33.3.1    DateTime.GetNow() and DateTime.GetNowUtc() .....	146
33.3.2    DateTime.Parse (“4/12/2019”) .....	146
33.3.3    DateTime.Add (years, months, days, hours, minutes, seconds)	147
33.3.4    DateTime.Set (year, month, day, hour, minute, seconds) .....	148
33.3.5    DateTime.Subtract (datetime) .....	148
33.3.6    Hour, HourDecimal, Minute, Second, Year, Month, MonthName, Day, DayOfWeek, and DayOfYear properties .....	148
33.3.7    DateTime.Date, DateTime.Time and DateTime.TimeHHmm....	149
33.3.8    DateTime.Iso8601 and DateTime.Rfc1123 .....	149
33.3.9    DateTime.AsTotalSeconds .....	149
33.3.10    Example of common properties.....	150
33.4    File Extension.....	150
33.4.1    file=File.AppendPicker(), file.AppendLine, file.AppendText() and file.Size	151
33.4.2    file=File.ReadPicker(“.txt”), file.ReadAll(), file.ReadLines().....	152
33.4.3    file=File.WritePicker(), file.WriteLine, file.WriteAllText() and file.Size	153
33.5    Gopher Extension for Gopher-of-Things .....	154
33.5.1    Gopher callback functions and Gopher menus.....	154
33.5.2    Common menu entry types .....	155
33.5.3    Selector versus ids.....	156
33.5.4    Gopher.AddRoute (“route”, “function”) .....	157
33.5.5    Gopher.Start (“name”).....	157
33.6    Http Extension .....	158
33.6.1    Http.Get(url, [headers]) .....	158
33.6.2    Http.Post() and Http.Put().....	159
33.7    Math Extension .....	160
33.7.1    Trigonometry (Math.Sin (radians) and more).....	160

Guide to Using Best Calculator	Page   11
33.7.2    Rounding and sign (Floor(), Round() and more) .....	161
33.7.3    Bit functions Math.BitAnd, Math.BitOr, and Math.BitNot .....	163
33.7.4    Logarithm and power functions (Math.Log, Math.Exp, and more)	
163	
33.7.5    Math.Factorial and Math.IsNaN.....	164
33.7.6    Math.PI, Math.E and Math.NaN values .....	165
33.7.7    Math.Fft and Math.InverseFft Fourier transform .....	165
33.7.8    Math.Sigma (x) .....	165
33.8    Memory Extension .....	166
33.8.1    Memory[<expression>] and Memory.<name> .....	166
33.8.2    GetOrDefault and IsSet functions .....	167
33.8.3    Memory technical details.....	168
33.9    Screen Extension .....	170
33.9.1    Screen.ClearLine(<line>) and Screen.ClearLines(<from>, <to>) .....	170
33.9.2    Screen.RequestActive() and Screen.RequestRelease().....	170
33.9.3    Screen.H and Screen.W Extension .....	171
33.9.4    Screen.GX and Screen.GY.....	172
33.10    Screen.Graphics() and Screen.FullScreenGraphics() Extension .....	174
33.10.1    Border = color .....	174
33.10.2    Clear() .....	174
33.10.3    GraphXY(data).....	175
33.10.4    GraphY(data, [name]) .....	176
33.10.5    SetPosition(X,Y) and SetSize(H, W) .....	177
33.10.6    SetMoved(f, arg), SetPressed(f, arg), SetReleased (f, arg)....	177
33.10.7    Updating Data with graph.Update() and PAUSE.....	179
33.11    Screen.Graphics Objects (Arc, Button, Slider, Text and more) .....	181
33.11.1    Common fields X1 Y1 X2 Y2 CX CY Opacity Rotate Data .....	181
33.11.2    Common methods: obj1.Intersect(obj2) .....	182

Guide to Using Best Calculator	Page   12
33.11.3    Arc (cx, cy, innerRadius, outerRadius, ang1, ang2).....	183
33.11.4    Button (x1, y1, x2, y2, "text", "function") .....	183
33.11.5    Circle(X, Y, radius [,yradius]) Line(X1, Y1, X2, Y2) Rectangle(X1, Y1, X2, Y2)	183
33.11.6    LineTo (x, y), MoveTo (x, y), and ClearGoTo() .....	185
33.11.7    Polygon() and poly.AddPoints() and SetPoints () .....	185
33.11.8    Slider (x1, y1, x2, y2, "text", "function") .....	186
33.11.9    Text (x1, y1, x2, y2, "text", fontsize) .....	188
33.12    Screen.Graphics Scaling (Advanced) .....	188
33.13    Sensor Extension .....	190
33.13.1    Using Sensor.Camera and making images .....	190
33.13.2    The simplest camera program .....	190
33.13.3    Using Analyze to modify the Sensor.Camera image .....	191
33.13.4    Example: HTML color from Sensor.Camera .....	192
33.13.5    Sensor.Compass (function) .....	193
33.13.6    Sensor.Inclinometer(function) .....	193
33.13.7    Sensor.Light(function).....	193
33.13.8    Sensor.Location Extension .....	194
33.13.9    Sensor.Microphone Extension .....	194
33.13.10    Full Sensor example .....	195
33.14    String Extension .....	197
33.14.1    String.Escape ("color", red, green, blue).....	198
33.14.2    String.Escape ("csv", <string or array>) .....	198
33.14.3    String.Escape ("json", <string or array>).....	199
33.14.4    String.Parse("csv", <data string>) .....	200
33.14.5    String.Parse ("json", <data string>) .....	201
33.14.6    String.Pos (str, lookFor, startingIndex) .....	202

Guide to Using Best Calculator	Page   13
33.14.7     String.Replace (string, replacement, startIndex) .....	202
33.14.8     String.ToUpper(str) and String.ToLower(str) .....	202
33.15       System Extension .....	203
33.15.1    System.Errors .....	203
33.15.2    System.FolderBasic() .....	203
33.15.3    System.FolderTemporary() .....	203
33.15.4    System.SetInterval (function, interval, argument) .....	203
33.15.5    System.Trace (0=off 1=normal).....	204
33.15.6    System.Version.....	204
34        Bluetooth Programming with Best Calculator, IOT edition .....	205
34.1      Programming Bluetooth using BC BASIC .....	205
34.2      Initializing your device and available properties .....	206
34.2.1    Error handling and Bluetooth .....	208
34.3      The <i>objects</i> you use when programming your Bluetooth device .....	209
34.3.1    The Bluetooth object .....	209
34.3.2    The Bluetooth.Devices object (Array / ObjectValueList) .....	210
34.3.3    Individual Bluetooth devices from Bluetooth.Devices.....	210
34.3.4    Specializations.....	211
34.4      Rfcomm (Serial-port) Bluetooth .....	212
34.5      Bluetooth.Watch (“specialization”, “function”) .....	213
34.5.1    Watch (“Bluetooth”, function).....	213
34.5.2    Watch (“Eddystone”, function).....	213
34.5.3    Watch(“Eddystone-URL”, function) .....	214
34.5.4    Watch (“RuuviTag”, function) .....	214
34.6      Selecting a device with PickDevicesNames and more .....	215
34.6.1    Bluetooth.PickDevicesName(<name pattern>) .....	215
34.6.2    Bluetooth.DevicesName(<name pattern>).....	216
34.6.3    Bluetooth.Devices().....	216

Guide to Using Best Calculator	Page   14
34.7    Reading data from raw Bluetooth services and <i>characteristics</i> .....	217
34.7.1    Direct device Read routines .....	218
34.8    Using callbacks to read data .....	220
34.9    Using the <i>specializations</i> for specific devices .....	222
35    Bluetooth Specializations for specific devices .....	223
35.1    Working with Arduino .....	224
35.2    Ardudroid Protocol.....	225
35.3    BBC micro:bit.....	227
35.4    beLight CC2540T Light development kit.....	230
35.5    DOTTI device.....	231
35.6    Hexiwear wearable platform.....	234
35.7    Infineon DPS310 .....	236
35.8    MagicLight and Flux light.....	238
35.9    MetaWear MetaMotion R device.....	239
35.10    NOTTI device .....	244
35.11    Puck.Js .....	247
35.11.1    Common Puck.Js JavaScript commands.....	247
35.11.2    Simplest puck.js program.....	248
35.11.3    A magnetometer program for the puck.js device.....	249
35.12    RuuviTag.....	251
35.12.1    Callback details.....	251
35.13    Skoobot .....	252
35.13.1    Skoobot Bluetooth commands .....	253
35.13.2    The simplest complete Skoobot program.....	254
35.13.3    Example: complete program to graph Light data .....	255
35.14    Slant Robotics LittleBot .....	257
35.15    TI SensorTag 2541 (original version) .....	259
35.16    TI SensorTag 1350 (2016 version) .....	262
35.17    TI SensorTag 2650 .....	267
35.18    TI Display (Watch) DevPack.....	268

	Page   15
Guide to Using Best Calculator	
35.18.1	268
35.18.2	268
35.18.3	268
35.18.4	268
35.18.5	268
35.18.6	269
35.18.7	269
36	Complete Examples.....
36.1	270
36.1.1	Connecting to Microsoft Flow .....
36.1.2	Set up the flow at Microsoft Flow.....
36.1.3	Write the BC BASIC program.....
36.2	Hiking with an altimeter .....
36.2.1	The Graph program.....
36.2.2	The Altitude program.....
36.2.3	The TARE program .....
36.3	The Happy Birthday Program .....
36.4	10 PRINT CHR\$(205.5+RND(1)) ;: GOTO 10 .....
37	Text and BC BASIC.....
37.1	Fixed character screen commands.....
37.2	Console Commands.....
38	Using the library, step by step .....
38.1	Add a new package for your program .....
38.2	Add a new program to your package .....
38.3	Edit the program to do the conversion .....
38.4	Run the program to test it.....
38.5	Bind the program to a key.....
38.6	Next steps.....
39	Appendix: Release Notes .....
39.1	Release Notes: 3.20 May 2019 .....

Guide to Using Best Calculator	Page   16
39.2    Release Notes: 3.19 May 2019 .....	298
39.2.1    Screen.GX and Screen.GY graphics sizes.....	298
39.2.2    Skoobot robot support.....	298
39.3    Release Notes: 3.18 April 2019.....	298
39.3.1    IOT version is now free! .....	298
39.3.2    Calculator COPY and PASTE.....	299
39.3.3    BC BASIC UI Import and Copy improvements .....	299
39.3.4    Variable names can include more characters .....	299
39.3.5    New Data object.....	301
39.3.6    DateTime improvements .....	301
39.3.7    DateTime.Parse (string) .....	301
39.3.8    DateTime.Add (years, months, days, hours, minutes, seconds)	301
39.3.9    DateTime.DayOfYear.....	301
39.3.10    DateTime.HourDecimal.....	301
39.3.11    DateTime.Set (years, months, days, hours, minutes, seconds)	301
39.3.12    New Gopher object to make a Gopher of Things server .....	301
39.3.13    Graphics.ClearGoTo .....	302
39.3.14    Graphics.YAxisMax and Graphics.YAxisMin .....	302
39.4    Release Notes (3.17, September 2018) .....	302
39.5    Release Notes (3.15, February 2018) .....	308
39.6    Release Notes (3.14, December 2017) .....	308
39.7    Release Notes (3.13, November 2017).....	309
39.8    Release Notes (3.12, November 2017).....	310
39.9    Release Notes (3.11, November 2017).....	311
39.10    Version 3.10, October 2017 Release Notes .....	311
39.11    Version 3.8, September 2017 Release Notes .....	312
39.12    August 2017, second version Release Notes .....	312
39.13    August 2017 Release Notes.....	313

	Page   17
Guide to Using Best Calculator	
39.14    July 2017 Release Notes.....	314
39.15    Spring 2017 Release Notes.....	314
39.16    Release Notes (3.15, January 2018) .....	315
39.17    Release Notes (3.17, September 2018).....	316
39.18    Release Notes (3.18, May 2019) .....	321
39.18.1    IOT version is now free! .....	321
39.18.2    Calculator COPY and PASTE.....	321
39.18.3    BC BASIC UI Import and Copy improvements .....	321
39.18.4    Variable names can include more characters.....	321
39.18.5    Array SetProperty (name, value) .....	321
39.18.6    New Data object.....	322
39.18.7    DateTime.Parse (string) .....	322
39.18.8    DateTime.Add (years, months, days, hours, minutes, seconds)	
322	
39.18.9    DateTime.DayOfYear.....	323
39.18.10    DateTime.HourDecimal.....	323
39.18.11    DateTime.Set (years, months, days, hours, minutes, seconds)	
323	
39.18.12    Graphics.ClearGoTo .....	323
39.18.13    Graphics.YAxisMax and Graphics.YAxisMin .....	323
39.19    Release Notes (3.19, May 2021) .....	323
39.19.1    Screen.GX and Screen.GY.....	323
39.19.2    Skoobot is now supported!.....	325
39.20    Release Notes (3.20, December 2021).....	327
39.20.1    Unicode tables are updated to the latest standard .....	327
39.20.2    INKEY\$ is initialized correctly .....	327
39.20.3    STOP SILENT command .....	327
39.20.4    Gopher.Start(..., port) .....	327

Guide to Using Best Calculator	Page   18
39.20.5 Skoobot updates!	327
40 Appendix: Sample programs	331
40.1 BT: An Overview of Bluetooth	331
40.1.1 List Bluetooth devices	331
40.1.2 Pick a Bluetooth device	332
40.1.3 Power	332
40.2 BT: BBC Microbit	333
40.2.1 Accelerometer	333
40.2.2 Button	334
40.2.3 Compass	335
40.2.4 Magnetometer	336
40.2.5 SetLed	337
40.2.6 Status	338
40.2.7 Temperature	338
40.2.8 Write (text, speed)	340
40.3 BT: beLight	340
40.3.1 Green	340
40.3.2 Red	341
40.4 BT: Dotti	342
40.4.1 An introduction	342
40.4.2 Change Mode	342
40.4.3 List BT Devices	343
40.4.4 Load screen from memory	344
40.4.5 Raw Bluetooth commands	345
40.4.6 SetAnimationSpeed	346
40.4.7 SetColumn and SetRow to random lines	347
40.4.8 SetName of a Dotti device	348

Guide to Using Best Calculator	Page   19
40.4.9    SetPixel to random green dots.....	348
40.4.10   SetPixel to write a single green dot .....	349
40.4.11   Sync Time .....	349
40.5      BT: Hexiwear.....	350
40.5.1   Accelerometer.....	350
40.5.2   Compass .....	351
40.5.3   List Information.....	352
40.5.4   Raw Access to Hexiwear .....	353
40.5.5   Read All .....	354
40.5.6   Set notification count.....	356
40.5.7   SetTime .....	357
40.6      BT: MagicLight .....	357
40.6.1   Green.....	357
40.6.2   Off .....	358
40.6.3   On.....	358
40.6.4   Red .....	359
40.7      BT: SensorTag 1350 .....	359
40.7.1   Accelerometer Gyroscope and Magnetometer .....	360
40.7.2   Accelerometer Off.....	361
40.7.3   Barometer .....	362
40.7.4   Button .....	363
40.7.5   Humidity.....	364
40.7.6   IO .....	365
40.7.7   IR .....	366
40.7.8   Optical Sensor .....	367
40.8      EX: BC BASIC Quick Samples.....	368
40.8.1   Colorful Countdown.....	368

	Page   20
40.8.2    Grams of Fat to Calories.....	369
40.8.3    Miles per Gallon .....	370
40.8.4    Right Triangle calculator .....	371
40.8.5    Tip Calculator .....	371
40.8.6    Welcome to BC BASIC .....	373
40.9     EX: Files, CSV and JSON, HTML, Flow .....	373
40.9.1    Appending to a file .....	373
40.9.2    Http.Get(url, headers) reads data from the internet.....	374
40.9.3    Microsoft Flow example.....	375
40.9.4    Read Entire File .....	378
40.9.5    Reading a CSV file.....	379
40.9.6    Writing to a file (including CSV) .....	380
40.10     EX: Financial .....	381
40.10.1    Common Tip Values .....	381
40.10.2    Compound Interest .....	382
40.10.3    Doubling Time .....	383
40.10.4    Future Value.....	385
40.10.5    Money Conversion .....	386
40.10.6    Present Value.....	387
40.10.7    Return on Investment .....	388
40.11     EX: Gopher and Gopher-of-Things .....	388
40.11.1    A simple GOPHER Program .....	389
40.11.2    GOPHER of things.....	390
40.11.3    Gopher program with changing responses.....	391
40.12     EX: Real Estate.....	393
40.12.1    Acres to square feet .....	393
40.12.2    Debt to Income calculations .....	393

Guide to Using Best Calculator	Page   21
40.12.3     Minimum and Maximum density.....	394
40.12.4     Rectangle in feet to acres.....	395
40.12.5     Square feet to acres .....	395
40.13       EX: Space and Astronomy .....	396
40.13.1    Arc Length .....	396
40.13.2    AU to Meters.....	397
40.13.3    Conversion Library .....	397
40.13.4    Distance to horizon .....	399
40.13.5    Lightyears to Parsecs.....	399
40.13.6    Meters to AU .....	400
40.13.7    Parsecs to Lightyears.....	400
40.13.8    Rocket Equation .....	401
40.14       EX: Statistics .....	402
40.14.1    Finite Population Correction .....	402
40.14.2    Margin of Error.....	403
40.14.3    Pfail.....	403
40.14.4    Sample Size .....	404
40.14.5    Sample Size Library .....	406
Index to Best Calculator Manual.....	409

There are two editions of Best Calculator: the classic Best Calculator which has been shipped, free and with no ads, for several years. It's extended to be programmable in BASIC. This simple language lets anyone write small programs to help automate some of the specialized math.



The standard edition splash screen with yellow "Programmable" stripe

The Best Calculator, IOT edition takes the familiar and powerful Best Calculator and adds the ability to connect to a range of IOT (Internet of Thing) devices. All the parts in common with the standard edition of Best Calculator are free; the rest require a small payment.

As of December, 2016, the IOT capabilities are the ability to communicate to Bluetooth LE devices. Both low level access and simplified, higher-level *specializations* are available for a number of Bluetooth devices. As of version 3.18, the IOT edition is free to use.



The IOT edition with the blue Bluetooth stripe

This manual describes both editions. There will be a note for each feature that is only part of a specific edition.

# 1 A QUICK TOUR OF BEST CALCULATOR

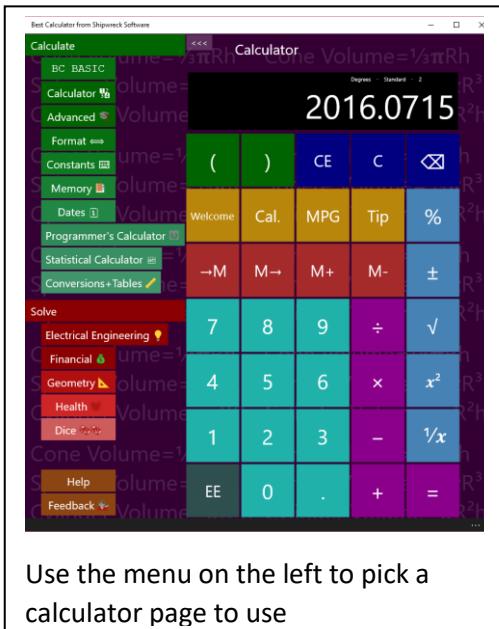
Best Calculator has *Pages* (in green) for common calculations, *Solvers* (in red) for specialized calculations, and Help and Feedback keys (in brown).

The most common pages that you'll use in Best Calculator are the *Calculator* and *Advanced* pages. The Calculator page (shown in the example) is a simple calculator.

The Advanced page is where you can access the angle (trigonometry), logarithm and other advanced math functions. When you are on a wide screen, selecting Advanced will display both the advanced functions and the regular calculator.

The available pages are:

1. BC BASIC lets you program your calculator using the popular BASIC language.
2. Calculator is a common calculator
3. Advanced includes trigonometry and logarithm functions
4. Format lets you change how the results are displays
5. Constants lists common physical constants (like gravity)
6. Memory lets you save, recall and change memory slots
7. Programmer's calculator is for hex and binary operations
8. Statistical calculator performs basic statistics on columns of numbers
9. Conversions and tables lets you convert between common units like converting liters to gallons. It also includes an ASCII table and Unicode lookup



The available solvers are

1. Electrical Engineering
  - a. Voltage, Current and Resistance ( $V=IR$ )
  - b. Resistors
  - c. Resistor color code
  - d. Capacitors
2. Financial
  - a. CAGR (Compound Annual Growth Rate)
  - b. Mortgage calculations
  - c. WACC (Weighted average cost of capital)
3. Geometry
  - a. Circles
  - b. Right Triangles
  - c. Slope
4. Health
  - a. Ideal Heart Rate
  - b. BMI (Body Mass Index) and BMI for Kids
  - c. Pulse
5. Dice

The feedback page lets you give feedback about Best Calculator. We always include requested features in each new release!

**Are you using a Microsoft keyboard?** You can program in the Best Calculator as the calculator program. See the Advanced Windows Features chapter for details.

**Back Button on the Phone.** The Back button lets you quickly switch between any two calculator pages or solvers. Once in a page, pressing back button once shows the menu. Pressing back button again switches you to the page you were at before.

For example, from the menu, go to the Calculator page. Tap the back button, and you are at the menu. Tap the Advanced page to go to the advanced page. Tap the Back button again to get to the menu, and once more to get to the Calculator page.

## 2 NUMBERS AND COMMON CALCULATIONS

### 2.1 SIMPLE ARITHMETIC

A room is 15 feet by 20 feet. How many square feet of carpet is required to cover the floor?

**Key in:**  $15 \times 20 =$

**Answer:** 300

Pressing the = key gives the answer to the entered formula. The calculator includes the normal + - × ÷ = math calculations.



Addition, Subtraction, Multiplication, Division, Equals, Entering numbers

### 2.2 CHAIN CALCULATIONS

Press the = key again to repeat the last calculation ( $\times 20$ )

**Key in:**  $15 \times 20 = =$

**Answer:** 6000

### 2.3 ALGEBRAIC ENTRY AND PARENTHESES

Calculations are performed as they are entered (“chain input”, or the confusingly-named “Algebraic entry”)

**Key in:**  $2 + 3 \times 4 =$

**Answer:** 20

The  $2 + 3$  is calculated first and that result is multiplied by 4; this is different from “school” arithmetic where multiplication and division are calculated first.

You can force the order of operations by using the left and right parenthesis keys.

**Key in:** 2 + ( 3 × 4 ) =

**Answer:** 14

The calculator will empty the display when you type the left parenthesis, will display the partial calculation (12) when you type the right parenthesis, and the final result when you type the equals sign (=) .

Best Calculator doesn't limit how deeply nested the parentheses are.

Best Calculator's Equation Input mode uses operator precedence where  $=1+2\times3$  is calculated as 7. This is similar to what some calculators call an "Algebraic Entry System with Hierarchy", "Algebraic Operating System", "Direct Algebraic Logic" or "Visually Perfect Algebraic Method".

## 2.4 EDITING ERRORS AND CLEARING THE DISPLAY

The C, CE and  $\square$  keys are used to edit input errors and clear the calculator. The Clear AI key clears the calculator to its default state.

**Key in:** 22 + 33 = C

**Result:** 0

**Key in:** 22 + 33 CE 44 =

**Result:** 66

The CE (Clear Entry) key clears the current entry. In the example, it clears just the '33' entry. When you type in '34' and '=', the calculation finished as if you had just entered 22 + 4 4. The answer is '66'.

**Key in:** 22 + 33  $\square$  7 =

**Result:** 59

The  $\square$  (Delete) key deletes just the last number entered. In the example, it clears just the second '3' in the '33' entry. When you type in '7' and '=', the calculator finishes its calculation as if you had just entered 22 + 37. The answer is 59.

## 3 MEMORY KEYS

### 3.1 MEMORY OPERATION

The four memory keys let you save and recall values from the calculator.

Save a number to memory

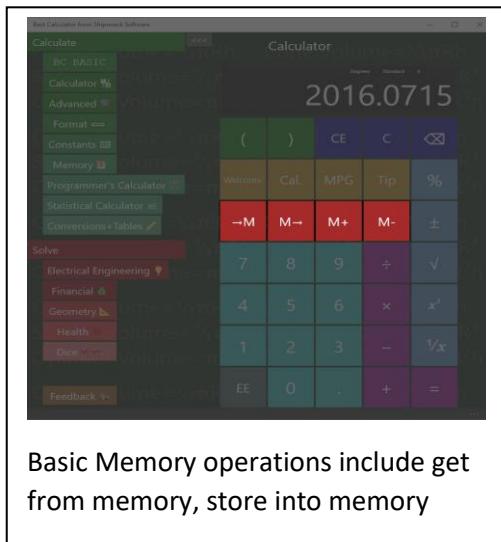
**Key in: 45 →M**

**Answer: 45 is displayed**

Recall a number from memory

**Key in: C M→**

**Answer: the screen is cleared and then 45 is displayed**



Basic Memory operations include get from memory, store into memory

The C key will clear the display. M→ will copy the memory value to the display.

See the Memory Page section for advanced memory usage.

### 3.2 EXAMPLE

You are planning on doing a number of calculations using your local tax rate (in the example, 8.25%). Store the tax rate into memory and then use it later on.

Calculate the tax (8.25%) on two different values (10 and 20)

**Key in: 8.25 →M**

**Key in: 10 + M→ % =**

**Answer: 10.825**

**Key in: 20 + M→ % =**

**Answer: 21.65**

The →M key places the current number into memory. The M→ key retrieves the memory value and places it into the display just as if you had typed it.

### 3.3 MEMORY ADD AND SUBTRACT

The M+ key adds and the M- key subtracts the current number to or from the existing memory value.

---

**Key in:** 45 →M M+ M→

**Answer:** 46

---

First the number 45 is put into memory (45 →M). Then the memory value is incremented (M+). Finally, the incremented value is displayed (M→)

The M- key subtracts the current number to the existing memory value.

## 4 MORE MATH

### 4.1 SQUARE ( $x^2$ ) KEY

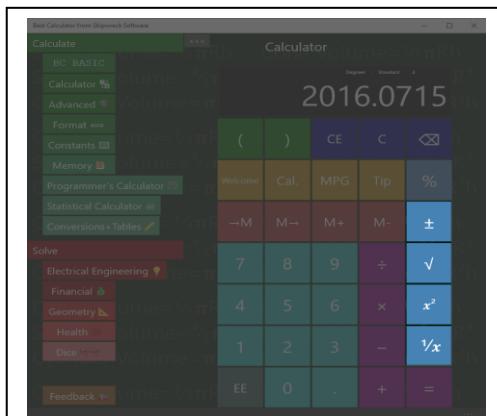
If you have a square 5 inches on a side, how many square inches is the square?

**Key in:**  $5 \times^2$

**Answer:** 25

The  $x^2$  key is the same as multiplying the current number by itself. In the example, it's the same as entering  $5 \times 5 =$

The  $x^2$  key squares the number instantly without you tapping the = key.



Best Calculator includes math functions like square root, square, inverse, change sign

### 4.2 SQUARE ROOT ( $\sqrt{ }$ ) KEY

The  $\sqrt{ }$  square root key finds the square root of the current number.

If you have a tile and know that it is 25 square inches, how long is it on a side?

**Key in:** 25  $\sqrt{ }$

**Answer:** 5

### 4.3 INVERSE ( $1/x$ ) KEY

What's  $\frac{1}{3} + \frac{1}{3}$ ?

**Key in:** 3  $1/x$  + 3  $1/x$  =

**Answer:** 0.666

### 4.4 CHANGE SIGN ( $\pm$ ) KEY

The  $\pm$  key will change the current number from positive to negative or from negative to positive.

## 5 SCIENTIFIC NOTATION

---

### 5.1 USING THE EE KEY

Some numbers are too large or too small to enter conveniently. This is where you can use the EE key to enter your number in exponential (scientific) notation.

Avogadro's number is about  $6.022 \times 10^{23}$  and the atomic weight of water is about 18. How many molecules of water are in a single gram of water?

---

**Key in:** `6.022 EE 23 ÷ 18 =`

**Answer:** `3.34 E 22, or  $3.34 \times 10^{22}$`

---

To enter a negative exponent, use the – key. The ± will change the whole number from positive to negative.

The mass of a single oxygen atom is about  $2.68 \times 10^{-26}$  kilograms. What is the mass of 20 atoms of oxygen?

---

**Key in:** `2.68 EE - 26 × 20`

**Answer:** `5.36E-25 or  $3.34 \times 10^{-25}$`

---

Best Calculator will display the result based on the selection in the Format section. If you are dealing with small numbers and are just seeing a “0” instead of a result in scientific notation, go to the Format screen and enter “Exponent”.

## 6 PERCENT KEY %

### 6.1 PERCENT (%) KEY

Given that you're already entered an equation (for example  $72 - 20$ ), pressing the % key will convert the 20 into 20% of 72.

If you have entered only a single number (e.g., just "5"), then the % key will convert the 5 into 0.05 (as if calculating 5% of 100)



The percent key is used for taxes and more

### 6.2 CALCULATING SALES TAX

You are buying an item that costs \$72 and the sales tax is 5%. What is the total cost?

**Key in:**  $72 + 5 \% =$

**Answer:** 75.60

### 6.3 CALCULATING SALES WITH A PERCENT DISCOUNT

You are buying an item with a 20% pre-tax discount, and the sales tax is 5%. What's the total?

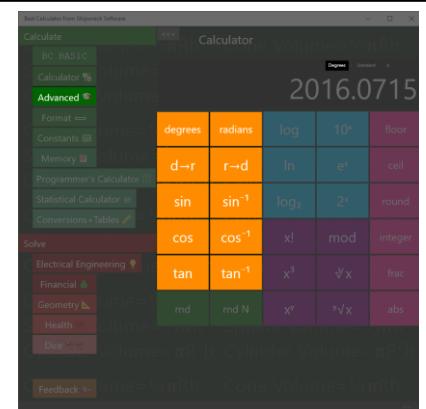
**Key in:**  $72 - 20 \% + 5 \% =$

**Answer:** 60.48

## 7 TRIGONOMETRY KEYS ON THE ADVANCED CALCULATOR

The trigonometry (angle) keys are part of the Advanced calculator.

Best Calculator includes sin, cos and tangent and their inverses.



Angle keys include sin, cos, tan, inverses, and degrees, radians and conversions.

### 7.1 CALCULATE IN DEGREES OR RADIANS

Calculations can be done in either degrees or radians. The display will show (in small type) whether you are currently in **degree** mode or **radian** mode.

Press the **degrees** key to switch to degree mode; press the **radians** key to switch to radian mode.

Press the **d→r** key to convert a number in degrees to a number in radians.  
Press the **r→d** key for the reverse conversion from radians to degrees.

Convert  $30^\circ$  to radians

**Key in:** 30 d→r  
**Answer:** 0.5236

The **d→r** and **r→d** keys work regardless of the **degrees** and **radians** settings.

### 7.2 SIN, COS, TAN

Calculate the sin of  $30^\circ$

**Key in:** 30 sin  
**Answer:** 0.50

If you get an answer of -0.9880, the calculator is in *radians* mode; key in **degrees** to switch to calculating in degrees

**Key in:** radians  $0.25 \times \pi = \cos$   
**Answer:** 0.7071

You can perform the same calculation using parenthesis

**Key in:** radians  $(0.25 \times \pi) \cos$   
**Answer:** 0.7071

The result of sin, cosine and tangent are always between -1 and 1.

### 7.3 INVERSE SIN, COS, TAN

(Also called arcsine, arccosine and arctangent)

Given the sin, cosine or tangent of an angle, you can get the original angle back out. The result will be an angle between  $0^\circ$  and  $360^\circ$  degrees or 0 to  $2\pi$  radians

Calculate the inverse sin (arcsine) of 0.5

**Key in:** 0.5  $\sin^{-1}$   
**Answer:** 30

If you get the answer 0.5236, the calculator is in radians mode; key in **degrees** to switch to degree mode.

The input values must be between -1 and 1; otherwise a NaN value is calculated.

The notation  $\sin^{-1}$  is the John Herschel notation; it means “inverse sine” and not “raised to a power”.

## 8 LOGARITHMS ON THE ADVANCED CALCULATOR

Logarithm keys are part of the advanced calculator.

Calculate the logarithm (base 10) of 100, 1000 and 100000,

**Key in:** 100 log

**Answer:** 2

**Key in:** 1000 log

**Answer:** 3

**Key in:** 100000 log

**Answer:** 5



Best Calculator includes logarithm and exponent keys

With base-10 logarithms, the log of a number is related to how many digits long the number is.

Best Calculator lets you calculate logs in three bases:

- The log key calculates using base 10
- The ln key calculates using base e (also called the natural logarithm)
- The log<sub>2</sub> key calculates using base 2 (binary, also called lb)

Each key is paired with the corresponding power key: 10<sup>x</sup>, e<sup>x</sup>, and 2<sup>x</sup>.

The base 2 logarithm is used by computer programmers to determine how many *bits* are required to hold a number of a certain magnitude.

How many bits are needed to hold a number that hold 26 distinct values?

**Key in:** 26 log2

**Answer:** 4.7004

## 9 POWERS, ROOTS, X!, MOD ON THE ADVANCED CALCULATOR

### 9.1 FACTORIAL (X!) KEY

Calculate  $6!$  (6 factorial)

**Key in:**  $6 \times!$

**Answer:** 720

$6!$  is another way of writing  $6 \times 5 \times 4 \times 3 \times 2 \times 1$



Powers and roots plus factorial and the mod operator.

### 9.2 MOD KEY

The Mod (Modulo) key calculates the remainder of a number. The remainder is the part that's left over when one number is divided by another.

What is the remainder of  $7 / 4$ ?

**Key in:** 11 Mod 4 =

**Answer:** 3

4 goes into 11 2 times with 3 left over.

### 9.3 CUBE ( $x^3$ ) AND CUBE ROOT ( $\sqrt[3]{x}$ ) KEYS

What is  $4.5^3$ ? (4.5 raised to the 3<sup>rd</sup> power, or  $4.5 \times 4.5 \times 4.5$  )

**Key in:** 4.5  $x^3$

**Answer:** 91.125

What is cube root of 27?

**Key in:** 27  $\sqrt[3]{}$

**Answer:** 3

## 9.4 ARBITRARY POWER ( $x^y$ ) AND ROOT ( $y\sqrt{x}$ ) KEYS

Best Calculator calculates numbers raised to arbitrary powers and take arbitrary roots. The powers do not have to be integers.

What is  $6^4$

---

**Key in:** 6 x<sup>y</sup> 4 =

**Answer:** 1296

---

What is the 4<sup>th</sup> root of 32?

---

**Key in:** 32 y<sup>v/x</sup> 4

**Answer:** 2.3784

---

# 10 ROUNDING AND ABS ON THE ADVANCED CALCULATOR

Rounding keys are in the advanced calculator.

**Floor:** round downwards to be a smaller number. The floor of a negative number (like -3.5) is rounded to a smaller number (-4)

**Ceil:** round upwards to a larger number. The ceil of a negative number (like -3.5) is rounded up to a larger number (-3)

**Round:** round towards to closest number. For example, 3.2 round is 3; 3.8 round is 4. Numbers that are exactly half-way between will round to the nearest even number (1.5 rounds to 2 and 4.5 rounds to 4)

**Integer:** rounds towards zero. The integer value of 4.5 is 4; and the value of -5.5 is -5. It is like removing everything past the decimal point.



Rounding, remainder and absolute value

(smaller) (larger)

The number line. Numbers further to the right are larger; numbers further to the left are smaller. -3 is smaller than -2 because it's further left.

**Frac:** returns the fraction part of a number

Calculate the remainder of 5.83

**Key in:** 5.83 remain

**Answer:** 0.83

**Abs:** convert negative numbers into positive numbers.

What is the absolute value of -4.5?

**Key in:** 4.5 +- abs

**Answer:** 4.5

# 11 RANDOM NUMBERS ON THE ADVANCED CALCULATOR

---

Best Calculator has two different random number keys.

The rnd key will place a random number between 0 and 1 into the result.

The rnd N key will place a random integer into the display; it will be between 1 and the number in the display.

Make a random number between 1 and 12

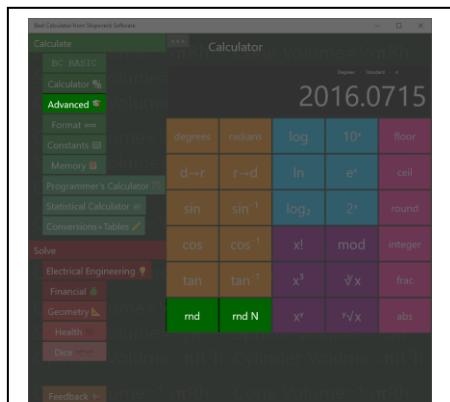
**Key in:** 12 rnd N

**Answer:** 3 (might be any number 1 to 12)

Make a random number between 0 and 1

**Key in:** rnd

**Answer:** 0.841 (might be any number 0 to 1)



Random Numbers

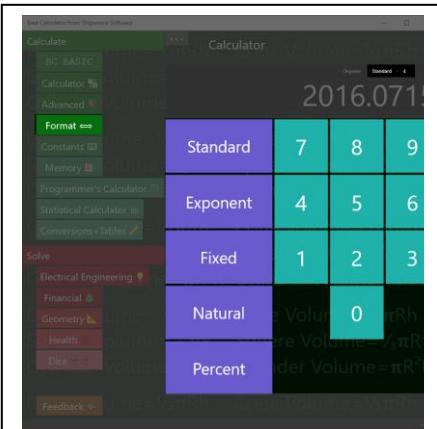
## 12 FORMATTING

Set how your results are displayed with the Format page.

Best Calculator can display numbers as regular numbers ("456.123") or as exponents ("4.56123E+002")

You can pick how many numbers after the decimal place to display.

The current formatting is always shown on the result display, right above the result. In the example, the display is "standard with 4 digits past the decimal point"



Change how your results are displayed with the Format page.

### 12.1 STANDARD FORMATTING

Regular number or exponent?	Automatically selected
Special features	"zero suppress" removes extra zeros after the decimal point.

The standard format is the one that's on by default when you first start the calculator. It will automatically switch between regular number and exponential form. The number of digits past the decimal place is a *maximum* number of digits; extra zeros are automatically suppressed (as is the decimal point).



In the example, up to 4 digits will be printed after the decimal point. The actual number (456.123000) has only zero beyond the ".123", and so they are suppressed.

## 12.2 EXPONENT FORMATTING

Regular number or exponent?	Always exponent
-----------------------------	-----------------

Sometimes called scientific notation, the exponent format is useful when you are dealing with large numbers much of the time.



In the example, the display (4.5612E+002) represents the number  $4.5612 \times 10^2$ . This in turn is  $4.5612 \times 100$ , or 456.12. The value is rounded from its real value (456.123) because the display has been set to only show 4 digits of precision.

A negative exponent (4.5612E-002) is less than zero (0.045612)

Exponent notation is equal to .NET's "E" format

## 12.3 FIXED FORMATTING

Regular number or exponent?	Regular number
Special features	Can overflow the display

The fixed format always displays using regular notation with a certain number of figures after the decimal point. It's often used when dealing with repeated calculations where each calculation should display the same way



In the example, exactly 4 digits will be printed after the decimal point.

If the number is too large to fit into the display, the display will be resized. After a certain point, the number will no longer fit, and will be truncated.

Fixed formatting is equal to .NET's "F" format

## 12.4 NATURAL FORMATTING

Regular number or exponent?	Regular number
Special features	Displays with commas

Similar to Fixed formatting, Natural formatting will display the number with commas separating the units.



If the number is too large to fit into the display, the display will be resized. After a certain point, the number will no longer fit, and will be truncated.

Fixed formatting is equal to .NET's "N" format

## 12.5 PERCENT FORMATTING

Regular number or exponent?	Regular number
Special features	Number is displayed as a percent: it's multiplied by 100 and displayed with a percent sign

The percent format is used when you need to show a number as a percent.



In the example, the number has been automatically multiplied by 100 and a percent sign is displayed. Note that the "real" number inside the calculator isn't changed: adding 1 to the example (456.123) results in 457.123 which is displayed as 457,12.3000%

Percent formatting is equal to .NET's "P" format

## 13 CONSTANTS

The Constants page lets to pick common physical constants often used in calculations.

### 13.1 $\pi$ AND e

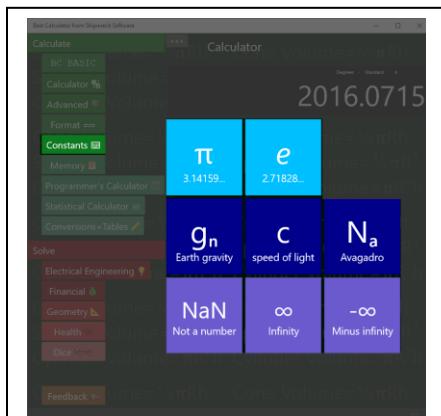
Press the  $\pi$  key to set the display to the value of  $\pi$  (about 3.1416).

Press the e key to set the display to the value of e (about 2.7183)

Calculate  $4 * \pi$

**Key in:**  $4 \times \pi =$

**Answer:** 12.5664



Best Calculator includes useful constants

### 13.2 $g_n$ , c AND $N_a$

$g_n$  (about 9.8) is the standard gravity in metric units; it's the gravitational acceleration on Earth, measured in meters/second<sup>2</sup>.

c is the speed of light (about 299792458) in meters/seconds

$N_a$  is Avogadro's number (about  $6.022 \times 10^{23}$ ) is the number of atoms or molecules in one *mole* of a substance. For example, there are  $N_a$  atoms of carbon (specifically, <sup>12</sup>C) in 12 grams of carbon.

### 13.3 NaN, $\infty$ AND $-\infty$

Best Calculator includes three special constants that can't be typed in otherwise.

NaN (Not a Number) means that a numeric value is not meaningful. For example, an inverse sin ( $\sin^{-1}$ ) is only meaningful for input values -1 to 1. Calculating  $\sin^{-1}$  of 2 will result in a NaN result.

$\infty$  is positive infinity, and  $-\infty$  is negative infinity.

## 14 MEMORY PAGE

The Memory page gives you access to 10 named memory slots. Each slot can be set, changed, and named.

The first memory slot (normally called Memory0) is the memory used by the main Calculator screen for memory operations.

### 14.1 SHOW THE OPERATION OF MEMORY0

The main calculator screen includes 4 memory keys ( $\rightarrow M$ ,  $M \rightarrow$ ,  $M+$  and  $M-$ ). These directly manipulate the first memory slot.

Enter 45 into memory slot 0. Go to the Calculator page.

**Key in:** 45 →

**Answer:** 45 is displayed

Now go to the Memory page. The first memory slot is set to 45.

### 14.2 SAVING AND ROAMING

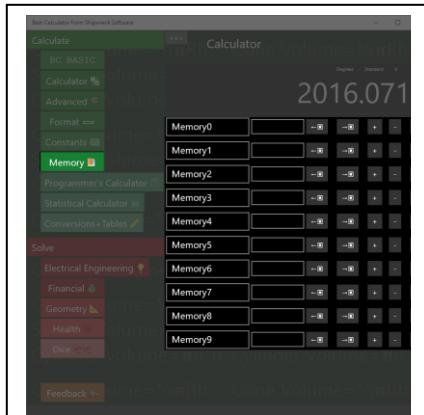
The memory values are roamed: when you set a memory value, the value and name are roamed to all of the computers where you're logged in with the same Microsoft Account. The values will be saved between calculator runs; you can exit the calculator and the values will be preserved for the next time you run Best Calculator.

### 14.3 NAME A MEMORY CELL

Click on the name of a memory cell to rename it.

Set, Increment, Decrement

Click on a memory value to change it. You should only enter numeric values!



Memory operations: rename, set, get from display, copy to display, increment, decrement, clear.

Click on the + key to increment a memory value

Click on the – key to decrement a memory value

#### **14.4 GET FROM DISPLAY AND COPY TO DISPLAY**

The  $\leftarrow\blacksquare$  key gets data from the display into the memory slot.

The  $\rightarrow\blacksquare$  key copies data from the memory slot to the display

#### **14.5 CLEAR**

The C key clears the memory slot

#### **14.6 MEMORY AND BC BASIC**

The values in the memory page are shared with the BC Basic Memory extension. You can get and set values from the calculator memory slots from within BC BASIC. Changes you make there will be reflected in the memory page.

## 15 DATES PAGE

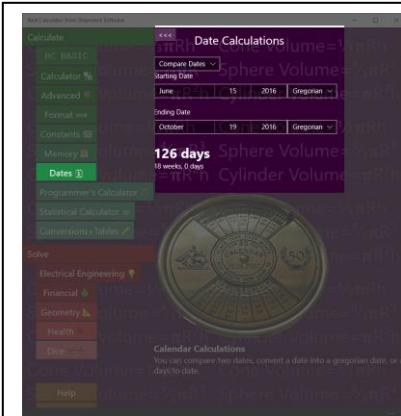
The Dates page lets you perform various date calculations.

### 15.1 COMPARE DATES

How many days are there between June 15<sup>th</sup> and October 19<sup>th</sup>?

**Key in:** Select Compare Dates from the drop-down. Set the Starting Date to June 15<sup>th</sup>, and the Ending Date to October 19<sup>th</sup>. Keep the Calendar type as Gregorian.

**Answer:** 126 days (18 weeks exactly)



Date calculations include calculating days between dates, adding days and calendar conversions

You can compare dates in multiple formats

- Gregorian is the standard calendar used in America and Europe
- Hebrew (הַלְוִת הָעֲבָרִי) is a lunisolar Jewish calendar
- Hijiri (گامشماری هجری خورشیدی) is a calendar used in Iran and Afganistan
- Japanese calendar
- Julian is the common calendar used in Europe and America until being replaced by the Gregorian calendar
- Korean
- Persian
- Taiwan
- Thai
- Umm Al-Qura

### 15.2 CONVERT TO GREGORIAN

The Great Feast of Theophany is celebrated on the 6<sup>th</sup> of January, Julian. In 2016, what day is this in the Gregorian calendar?

**Key in:** Select Convert to Gregorian from the drop-down. Set the Starting Date to January 6<sup>th</sup>, 2016 and set the calendar type to Julian.

**Answer:** January 19, 2016

### 15.3 ADD DAYS

A project is started on April 19, 2016, and will take 21 days. What day will the project end on?

**Key in:** Select Add days from the drop-down. Set the Starting Date to April 19<sup>th</sup>, 2016 and keep the calendar type as Gregorian. St the Number of days to add to 21.

**Answer:** Tuesday, May 10, 2016

### 15.4 SUBTRACT DAYS

A project needs to end on November 19, 2018 after taking 21 days. What day should the project start on?

Subtracting days is just like adding days, but with a negative amount.

**Key in:** Select Add days from the drop-down. Set the Starting Date to November 19<sup>th</sup>, 2018 and keep the calendar type as Gregorian. St the Number of days to add to 21.

**Answer:** Monday, October 29, 2018

# 16 HEX, DECIMAL, OCTAL, BINARY ON THE PROGRAMMER'S CALCULATOR

The Hex, Decimal, Octal and Binary entry and conversions are part of the Programmers Calculator.

## 16.1 SETTING THE MODE

Press the bin, oct, dec or hex keys

to switch to binary (base-2), octal (base-8), decimal (base-10) or hexadecimal (base 16). The display will show which mode the Programmer's Calculator is in.

To enter a value, simply press the keys 0 to 9 or A-F. Valid keys will be displayed in Cyan; invalid keys are gray. In the example, the calculator is in decimal mode so that keys 0 to 9 are all valid. In binary mode, only keys 0 and 1 are valid; in octal mode only keys 0 to 7, and in hexadecimal mode keys 0 to 9 and A to F.

Add the number 1A to the number 12

---

**Key in:** hex 1 A + 1 2 =  
**Answer:** 2C

---

## 16.2 CONVERTING BETWEEN BASES

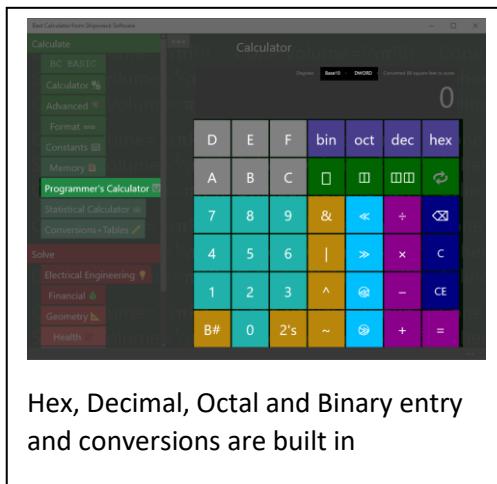
To convert from the current base to a different base, simply press the new base number. The existing display will be converted to the new base

Convert  $2C_{16}$  into decimal

---

**Key in:** hex 2 C dec  
**Answer:** 44

---



Hex, Decimal, Octal and Binary entry and conversions are built in

# 17 BIT OPERATORS ON THE PROGRAMMER'S CALCULATOR

The Bit operators are part of the Programmer's calculator

## 17.1 B# (COUNT BITS)

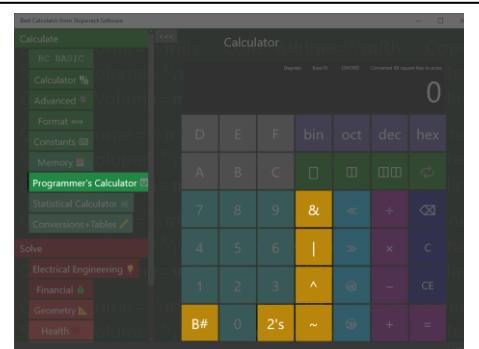
Use the B# key to count the number of '1' bits.

How many bits are '1' bits in the hex number 44?

**Key in:** hex 4 4 B#

**Answer:** 2

Hex 44 is binary 01000100. Only two of those bits are '1' bits.



Basic Memory operations: get from memory, store into memory

## 17.2 INVERSE (~) AND 2'S COMPLEMENT

The ~ key will invert each of the bits of the current value. This is also called the "1's complement" of the number.

The 2's key calculates the "2's complement" of a number. This is what (almost every) computer stores as the negative value of a number.

How does a computer store -1?

**Key in:** dec 1 2's

**Answer:** FFFFFFFF

## 17.3 AND (&), OR (|), XOR (^)

The And, Or and Xor keys will and, or xor (exclusive-or) two numbers.

What is 3 AND 4?

**Key in:** 3 & 4 =

**Answer:** 7

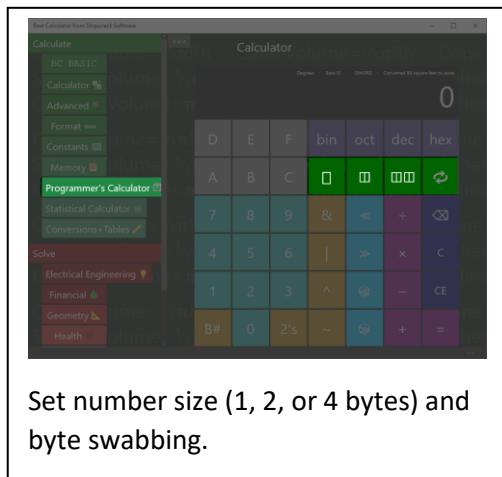
# 18 BYTES AND SWABBING

## ON THE PROGRAMMER'S CALCULATOR

The bytes and swap operators are part of the Programmer's calculator.

### 18.1 BYTES

Many programmer calculations change depending on the number of *bytes* that are involved in an operation. Common sizes are byte (□), word (2 bytes, □□) and dword (4 bytes, □□□□). The result display shows the number of bytes as “BYTE”, “WORD” and “DWORD”. Results that are larger than the current setting will be truncated.



Set number size (1, 2, or 4 bytes) and byte swabbing.

### 18.2 ⚡ SWAB (SWAP BYTES) KEY

The key performs the SWAB (swap bytes) operation. This is commonly used when dealing with network operations: most network protocols send data using “big endian” format while most PCs are in “little endian” format.

What's the network byte order for port 80? Port numbers are sent as 2 bytes.

**Key in:** □ dec 8 0 ⚡

**Result:** 20480

**Explanation:**

- □ switches to WORD (2 byte) mode
- dec switches to decimal mode
- 8 0 puts the number 80 into the display
- ⚡ (SWAB) switches the first and second byte around

# 19 SHIFT OPERATORS ON THE PROGRAMMER'S CALCULATOR

The Shift operators are part of the Programmer's calculator

The shift operators (and especially the rotate keys) use the byte setting.

$\ll$  is shift left

$\gg$  is shift right

$\ll\circlearrowleft$  is rotate left

$\gg\circlearrowright$  is rotate right

Shift hex F0 left by 2 bits

**Key in:** hex F 0  $\ll$  2 =

**Answer:** 3C0

In binary, F0 is 1111 0000. Shifted left 2 bits, the result is 11 1100 0000, which is 3C0. Bits introduced on the right are zero.

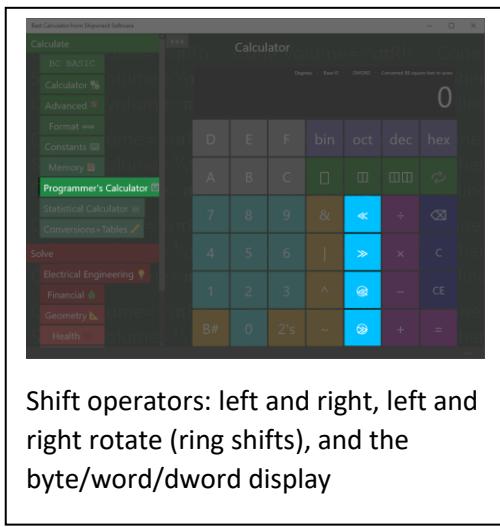
Ring-shift left F0 by 1 bit as a single byte

**Key in:** hex F 0  $\circlearrowleft$  1 =

**Answer:** E1

In binary, F0 is 1111 0000. With a normal shift, the bits that "shift out" will be dropped and new bits are zero. With rotate (ring shift), the bits would have been dropped are reintroduced on the other side.

In binary, F0 is 1111 0000. When rotated, the top bit (1) is put in as the lowest bit, resulting in 1110 0001, or E1.



# 20 PROGRAMMER'S

## MATH

The programmer's calculator can act as a regular calculator, but only for integers.

Divide 7 by 3

**Key in:**  $7 \div 3 =$

**Answer:** 2

Unlike the regular calculator, where  $7 \div 3$  is 2.3333, the Programmer's Calculator is strictly an integer-only calculator.

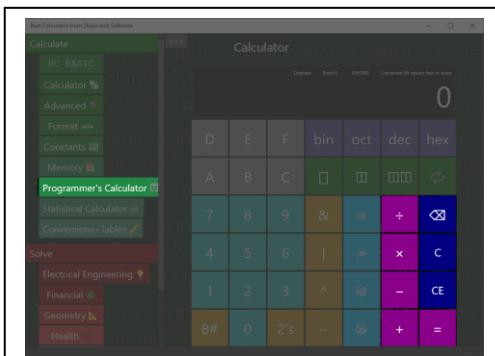
Additionally, the byte settings will force some results to be truncated

Multiply  $99 \times 99$  in byte mode

**Key in:**  $99 \times 99 =$

**Answer:** 73

In the regular calculator,  $99 \times 99$  would be 9801. This is  $(256 \times 38 + 73)$ . Only the bottom byte (the 73) is kept from the calculation. The rest is discarded.



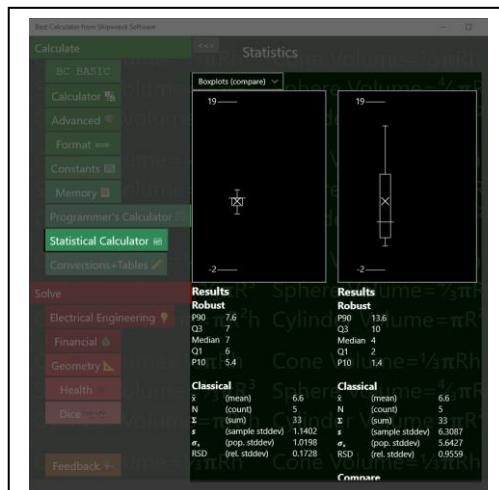
The programmer's calculator performs standard math operations

# 21 STATISTICAL CALCULATOR

The Statistical calculator takes in one or two lists of numbers and computes robust and classical statistics, T-tests and linear regression between two paired groups of numbers

The statistical calculator will display the data either as one or two boxplots (specifically, Tukey boxplots) or as an XY scatterplot.

Best Calculator always computes all statistics. As you type numbers into the data boxes Best Calculator computes and displays the results.



Statistical calculator showing the difference between two samples

## 21.1 PARTS OF THE STATISTICAL CALCULATOR SCREEN

The Statistical calculator screen is divided into three parts: the graphs, the computed results, and the data. There are two data entry boxes, a left box and a right box, resulting in a left hand data set and a right-hand data set.

1. **Graphs at the top** present a pictorial view of your data. You can choose the kind of graphs: two independent boxplots, two boxplots for comparison (their Y axes will be set to be the same), an XY Scatter plot, or no graph
2. **Computed results** for the left and right data sets. Both data sets will always compute Robust and Classical statistics. The right-hand data set in addition will compute a T-Test comparison between the left and right data sets and a linear regression between the left and right data sets.
3. **Left and right data sets** and their data entry boxes. The data is simply a list of numbers; you can copy and paste from Excel or simply type the data in.

## 21.2 ENTERING DATA

Enter data into the left and right data boxes.

When Best Calculator starts, each box is marked “Data” and contains some default values so you can see what kind of statistics and graphs Best Calculator will produce.

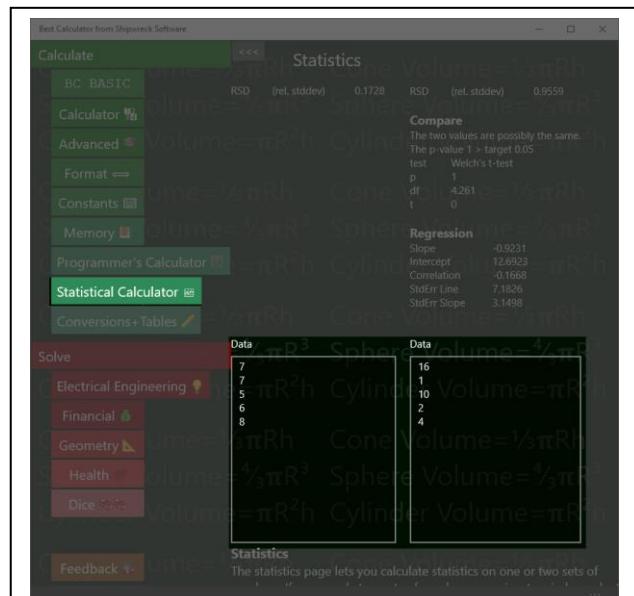
To enter data, simply click on either box. Data should be entered with one item per line. To enter a lot of data, you can paste the data into the text box.

Best Calculator is a calculator, not a spreadsheet, and does not understand file

formats like CSV (comma separate value) files, and does not have a way to read a file from disk.

You can paste values directly from Excel. In Excel, highlight and copy the column of numbers you want to compute statistics for. Then click on the text box in Best Calculator and paste.

Data which are not understood as numbers will be ignored. The first line with text in it will be the title of the data. For example, if you copy a column of numbers from Excel, you can include the column header and it will be used as the data title. As you enter data, Best Calculator will automatically recalculate your statistics.



Enter data either into the left or right text boxes. Statistics are automatically computed as you type.

The left box still has the default data in it; the right box has been cleared. The graphs have been turned off.

## 21.3 CLASSICAL STATISTICS (RESULTS)

Both the left and right hand data sets will compute classical statistics. Classical statistics work best with symmetrical, bell-shaped **normally** distributed data.

Lines that are text, not numbers, or which are blank are never included in the values.

### Classical

$\bar{x}$	(mean)	6.6
N	(count)	5
$\Sigma$	(sum)	33
s	(sample stddev)	1.1402
$\sigma_n$	(pop. stddev)	1.0198
RSD	(rel. stddev)	0.1728

### Mean

The mean ( $\bar{x}$ ) of the numbers is the arithmetic mean, or average of the numbers. It's calculated by adding up all the numbers and dividing by the count of the numbers.

### Count

The count (N) of the number of numbers in the data box.

### Sum

The sum ( $\Sigma$ ) is the total of all of the numbers added together.

### Sample Standard Deviation

The sample standard deviation (sample stddev, or s) is very similar to the population standard deviation, but the N value used is (N-1). The standard deviation informs you of the overall spread of the data.

### Population Standard Deviation

The population standard deviation (pop. stdev , or  $\sigma_n$ ) is the standard deviation assuming that the data in the data box is the entire population and not a sample. It's similar to but always a little smaller than the sample standard deviation.

### Relative Standard Deviation

The Relative standard deviation (RSD) is a normalized version of the sample standard deviation. It's the computed by dividing the sample standard deviation by the mean. It's useful for determining the "spread" of a sample without having to mentally compare the standard deviation with the mean.

**ROBUST STATISTICS (RESULTS)**

Both the left and right hand data sets will compute robust statistics. Robust statistics are less sensitive to outliers than classical statistics. The values computed by Best Calculator are used to display the Boxplot.

Lines that are text, not numbers, or which are blank are not included in the value.

**Robust**

P90	7.6
Q3	7
Median	7
Q1	6
P10	5.4

Computed robust statistics

The **median** value is often used as a representative measure of the data. When the data is symmetrical, the mean (classical) and median (robust) are the same; when data is skewed, the median represents a more typical member of the data while the mean is more weighted towards the high end.

To calculate the robust statistics, the data is sorted. Each robust data point is the value that is a certain percentile of the overall data. For example, the median is the 50% percentile; half of the data points are larger than the median value, and half smaller.

Point	Percentile	Comments
P90	90 <sup>th</sup>	The P90 measure is used to help estimate the spread of the data. In the boxplot, the P90 point is marked with a small circle.
Q3	75 <sup>th</sup>	The Q3 (third quartile) is the $\frac{3}{4}$ mark. The box in the boxplot is bounded by the Q# and Q1 points.
Median	50 <sup>th</sup>	The median (Q2) is taken from the exact middle of the sorted data set. It's marked by a long horizontal bar in the box of the boxplot.
Q1	25 <sup>th</sup>	The Q1 (first quartile) is the $\frac{1}{4}$ mark
P10	10 <sup>th</sup>	The P10 measure is the opposite of the P90 measure. In the boxplot, the P10 is also marked with a small circle.

## REGRESSION (RESULTS)

Use the regression data and the linear regression chart to tell if two data sets are related.

The data shows the thickness (in mils) of a layer of silver deposited onto a computer chip after a certain amount of time in a furnace.

The number of minutes is entered into the left hand data box; the thickness is entered into the right-hand data box. If both data sets have the same number of data points, Best Calculator will compute the linear regression statistics.

Values calculated are

**Slope** and **Intercept** is the best fit

for a single straight line through the data. These values can be placed directly into the standard formula for a line,  $y = mx + b$ . The slope value is the  $m$  value, and the intercept value is the  $b$  value.

The **correlation** coefficient says how close of a match the data is. If the data isn't correlated at all, the correlation is 0. A negative correlation means that as one value increases, the other decreases (also called a negative correlation). A value of 1 or -1 means that the data is perfectly correlated.

The **StdErr Line** (standard error of the line) and **StdErr Slope** (standard error of the slope) are tests of how noisy the data is and how good of a fit the computed slope and intercepts are.

## Regression

Slope	0.0037
Intercept	0.0094
Correlation	0.9814
StdErr Line	0.0007
StdErr Slope	0.0004

Computed linear regression statistics

Sample data:

Minutes	Thickness
1	0.0132
1.5	0.0151
2	0.0167
2.5	0.0177
3	0.0211

## 21.4 COMPARE WITH T-TESTS (RESULTS)

When two data sets are entered, Best Calculator will compute a Welch's t-test value.

Welch's t-test are used to decide if the two data sets are "the same" or "different". The boxplots are used informally for the same purpose.

### Compare

The two values are probably DIFFERENT.

The p-value 0.0145 is <= target 0.05

test Welch's t-test

p 0.0145

df 13.9359

t 2.7894

Computed robust statistics

Note that the t-test is not perfect way to tell if two samples are "different". For example, the numbers [10, 20, 30, 40] will be reported to be "possibly the same" as [25, 25] even though a person would declare them to be very different.

In the example, the two data sets are from two types of chemical analysis (<http://www.fao.org/docrep/w7295e/w7295e08.htm>)

The computed p-value helps answer the question, "are these two data sets likely to be from the "same" kind of data. If the p value computed is small (set to 0.05 in Best Calculator), the two data sets must not be the same and are therefore different. If the p value is large, then no conclusion can be drawn: perhaps the data is different, but perhaps it's not.

Welch's t-test is a more modern version of the Student's t-test. It's been shown to be as good as the Student's t-test when the sample variances are the same, and better than they aren't.

Although the primary calculation from the Welch's t-test is the p value, the df (degrees of freedom) and t statistic are also presented.

## 21.5 BOXPLOTS (GRAPH)

Boxplots are a simple way to visually compare two data sets.

The tick marks on the left (1.4 and 3 in the example) tell you about what the range of data is.

The central box shows the inter-quartile range (IQR) for the data. Exactly  $\frac{1}{2}$  of the data falls within the central box;  $\frac{1}{4}$  is less than and  $\frac{1}{4}$  is more than the box.

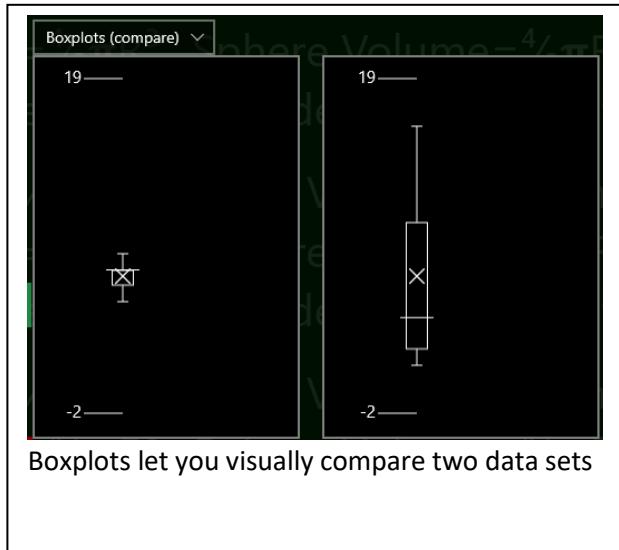
The horizontal line in the center of the box is the median of the data set.

The X in the boxplot (usually, but not always, somewhere in the box) is the average value. The X is the only part of the boxplot computed with classical statistics instead of robust statistics.

The whiskers can each be as long as  $1.5 \times$  the IQR. They are trimmed back to show the furthest-away data point that fits in the whisker. Outliers are not drawn.

The plots also shows the 10% and 90% percentile data points. These are useful when determining whether a process has a significant number of outliers or not. These are shown as the small circles, and will only be displayed when the data has at least 10 data points.

When you're displaying two data sets, you can **compare** the two boxplot or display them as **independent**. In compare mode, the two boxplots are display using the same scale. In independent mode, the two boxplots are display with their own scale.



## 21.6 XY SCATTER-

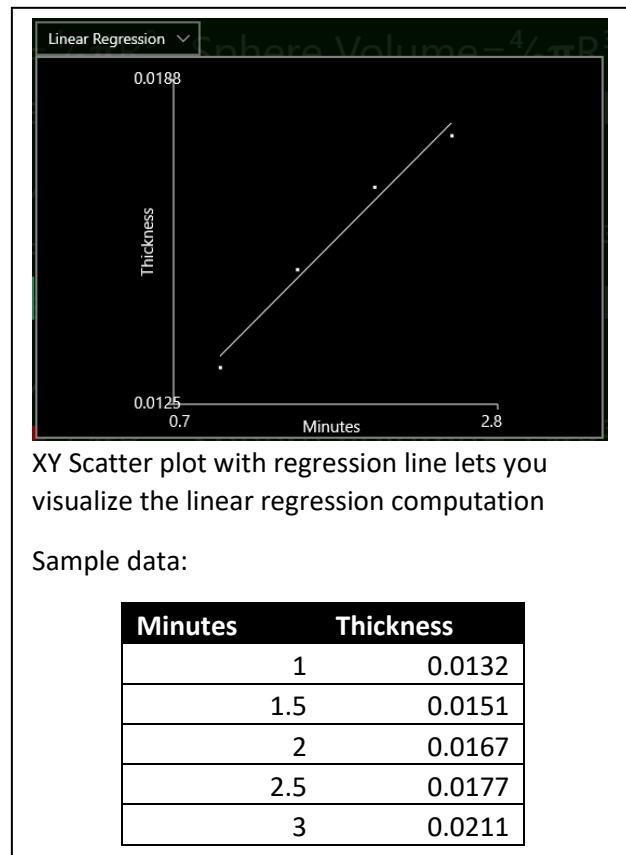
### PLOTS (GRAPH)

XY Scatterplots are a quick way to compare *paired* data.

The sample data used in the example is the same data used in the Regression example.

To make a scatterplot from your data, enter the *independent* data into the left hand data box and the *dependent* data into the right hand data box. Then chose “Linear Regression” as your graph type.

Each pair of data is plotted as a small dot, and the results of the regression analysis are drawn as a straight line. The headers will be displayed as the X and Y axis labels.



The min and max axis values are picked based on the data and are not settable.

Highly correlated data will be plotted close to the regression line; uncorrelated data will be plotted randomly, neither close to nor avoiding the regression line. The *correlation coefficient* is computed and displayed as part of the correlation results.

## 22 CONVERSIONS

Conversions (like inches to centimeters) are in the Conversions and Tables section.

Tap that section first, and then tap the conversion page you want.

### 22.1 CONVERTING

All of the converters work the same way.

**About Length**  
Inches, Feet, Yards and Miles are customary units of length in the United States, Canada, and other countries. There are exactly 25.4 centimeters per inch. There are 12 inches in a foot, 3 feet in a yard, and 1760 yards in one mile. There are 9 square feet in a square yard, and 27,878,400 square feet in a square mile.  
**Acres** are commonly used to measure land; there are 43,560 square yards in one acre.

Conversions include area, energy, length and more.

To convert a value, enter in the value you have (like “Square inches”) and read off from the resulting values.

Example: An apartment is 330 square feet. How many square yards is that?

**Key in:** tap ‘Conversions and Tables’ and then tap ‘Area’. Tap the text box next to ‘Square Feet’ and type in 330.

**Answer:** read off 36.67 from the Square yards box

As you enter a number, the surrounding values are automatically calculated.

How many acres are in a square mile?

**Key in:** tap ‘Conversions and Tables’ and then tap ‘Area’. Tap the text box next to ‘Square Miles’ and type in 1.

**Answer:** read off 640 from the Acres box

### 22.2 COPY TO AND FROM THE CALCULATOR RESULTS DISPLAY

You can copy a number from the calculator results display into any of the text boxes with the  $\leftarrow$  key that's next to each text box.

Your apartment is 10 yards by 4 yards. How many square feet is it?

**Key in:** in the main calculator, enter  $10 \times 3 =$

**Tap ‘Conversions and Tables’ and then tap ‘Area’.** Tap the  $\leftarrow$  key next to the text box marked ‘Square Yards’. It will be set to

360.

**Answer: read off 360 from the Square Feet box**

---

A square foot of carpet costs \$2.04. What is the cost to carpet your apartment?

---

**Key in: from the last example, tap the  $\rightarrow$  key next to the square feet box. Go to the main calculator; note that it has been set to 360. Key in  $\times 2.04 =$**

**Answer: \$734.40**

---

## 22.3 AREA CONVERSIONS

Best Calculator can convert between

- Square inches
- Square feet
- Square yards
- Acres
- Square miles
- Square centimeters, meters, hectares, and square kilometers.

A hectare is 10,000 square meters, or 1/100 of a square kilometer.

## 22.4 ENERGY CONVERSIONS

Best Calculator can convert between:

- Ergs
- Joules
- Kilowatt-Hours
- Calories
- Food calories (KCAL)
- Donuts
- BTUs
- Therms

## 22.5 LENGTH CONVERSIONS

Best Calculator can convert between

- Inches

- Feet
- Feet + Inches (output only)
- Yards
- Miles
- Centimeters
- Meters
- Kilometers

## 22.6 TEMPERATURE CONVERSIONS

Best Calculator can convert between

- Degrees Celsius (also called Centigrade)
- Fahrenheit
- Kelvin (absolute temperature in the metric system)
- Rankine (absolute temperature in Fahrenheit)

## 22.7 WEIGHT

Best Calculator can convert between

- Ounces
- Pounds
- Pounds + Ounces (output only)
- Short tons (2000 pounds)
- Long tons (2240 pounds)
- Grams
- Kilograms
- Tonnes (1000 kilograms; 2204 pounds)
- MMT (millions of metric tons)
- Grains
- Troy Ounces
- Troy Pounds
- Tolä (about 0.41 ounces)
- Sèr (80 Tolä)
- Maund (40 Sèr)

Guide to Using Best Calculator

## 22.8 FARM VOLUMES (US)

Page | 63

Best Calculator can convert between

- Cups
- Pints (2 Cups)
- Quarts (2 Pints)
- Gallons (4 Quarts)
- Pecks (1 Gallons)
- Bushels (4 Pecks)
- Liters

Given a weight in pounds per bushel, Best Calculator will also calculate the weight of an amount in bushels.

## 23 ASCII TABLE

The ASCII (American Standard for Computer Information Interchange) is a common way for computers to encode English-language text as numbers suitable for computer operation.

Programmers often need to convert from characters (like “@”) into their ASCII equal (the number 64).

The example shows the conversions for the “@” characters, sometimes called the “Ray” in honor of Ray Tomlinson, the creator of Internet email in the 1971.

The screenshot shows the 'ASCII Table' section of the Best Calculator software. The table is a grid where each row represents a character and its ASCII value. The columns are labeled with values 0 through 127. The first few rows show the following data:

Value	Character	Value	Character	Value	Character
0	0	1	@	P	~
1	!	2	1	Q	q
2	"	3	B	R	r
3	#	4	C	S	s
4	\$	5	D	T	t
5	%	6	E	U	u
6	&	7	F	V	v
7	(	8	G	W	w
8	)	9	H	X	x
9	*	:	I	Y	y
:	:	J	Z	j	z
+	:	K	[	k	{
,	<	L	\	l	
-	=	M	]	m	}
.	>	N	^	n	~
/	?	O	-	o	%

ASCII table

64 is the decimal value of @  
40 is the hex value of @  
100 is the octal value of @

In the example, the “@” is shown along with the decimal (64), hex (40) and octal (100) values.

Example: replace a character with its percent encoding (used in encoding URLs from arbitrary characters). The rules for percent encoding is to replace the single character with three characters: a “%” sign and then two digits with the character’s hex value. The @ hex value is 40, so to replace a single “@” in a URL, you need to replace it with “%40”.

# 24 UNICODE

Unicode is the worldwide specification for characters from all languages, including emoji and useful math symbols. Best Calculator supports Unicode 11.0, introduced in 2018

## 24.1 SEARCH RULES

Enter a set of search terms into the search box. As you type, the display is updated with matching characters.

A search term is a match if it's any part of the character description; short terms (1 or 2 characters long) and terms that start with = have to match a word exactly. Examples: a matches LATIN CAPITAL LETTER A and =phi matches LATIN SMALL LETTER PHI. Longer terms will match anywhere in a description.

Example: LATI matches LATIN CAPITAL LETTER A

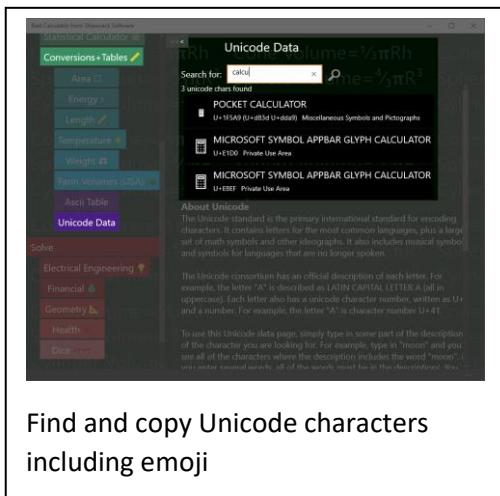
Terms that start with a minus sign (-) are anti matches; they invert the normal processing.

Terms that start with U+ will match a Unicode number exactly.

Each Unicode character includes its Unicode number (like U+41 for LATIN CAPITAL LETTER A), the character itself, the official description and the official alternate description. In addition, searches include the Unicode "Block" name. For example, IPA will match all of the characters in the IPA Extensions block.

## 24.2 COPYING CHARACTERS

Right-click on a result to bring up the app bar. Tap the clipboard to copy the character to the clipboard.

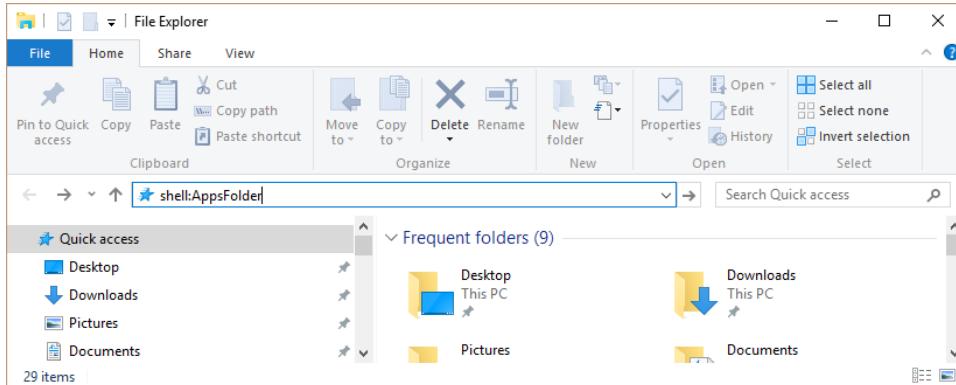


## 25 ADVANCED WINDOWS FEATURES

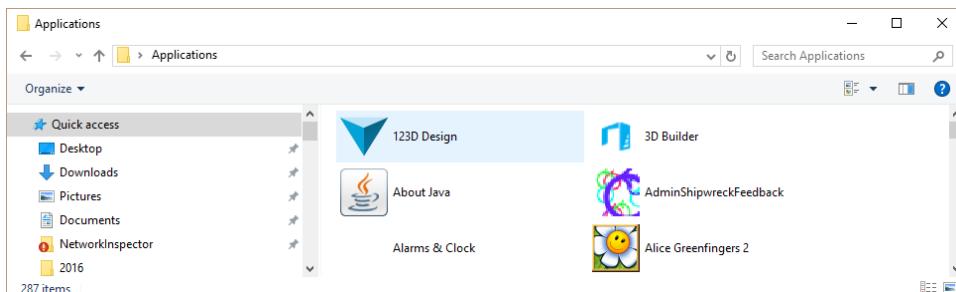
### 25.1 SHORTCUT ON THE DESKTOP

You can add Best Calculator as a shortcut on your desktop.

The easiest way is to start the regular Windows Explorer (press Windows-E). In the address bar, enter shell:AppsFolder (see the picture below).



The Explorer will show you all of your installed apps.



Right-click Best Calculator and select “Create a Shortcut”. You will be told that you can only create a shortcut on the desktop; click “Yes”. A shortcut to Best Calculator will be placed on your desktop.

## 25.2 SET AS THE CALCULATOR KEY

Windows keyboards often include pre-programmed buttons to launch common applications.

You can program the launch buttons to launch Best Calculators.

Run the Microsoft Mouse and Keyboard Center by pressing the Windows key and typing “Mouse and Keyboard”. The Mouse and Keyboard center program will show up.

In the example, the center is programming a Microsoft Wired Keyboard 600 with a Calculator key. We’re going to program the calculator key to start Best Calculator.



Under the Calculator setting, tap “Open a program, webpage, or file”. Then tap the Windows Explorer button and *carefully* enter

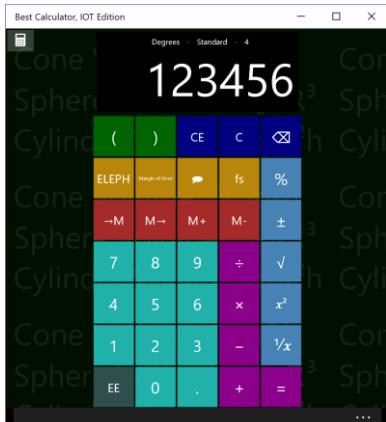
```
explorer shell:AppsFolder\48425ShipwreckSoftware.BestCalculator_jh2negtepkzpr!App
```

Then press the back button.

That’s it! The calculator button should now launch the calculator app. If you mis-type the long command line, the Windows Explorer will launch instead.

## 26 APPEARANCE AND ALIGNMENT

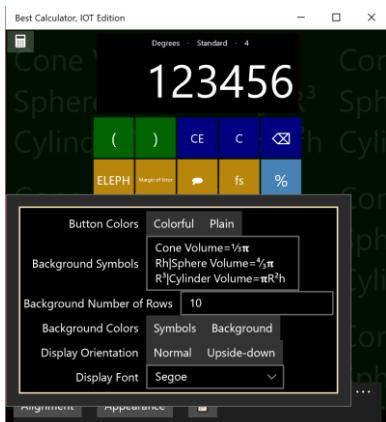
Best Calculator's appearance can be configured to suit your mood!



Colors colors and a modern typeface



Plain colors and 7-segment display



The appearance menu



Left aligned, using an antique segmented display

One of the niftiest features is that you can set the background to include formulae or text. The default text is

**Cone Volume= $\frac{1}{3}\pi Rh$  | Sphere Volume= $\frac{4}{3}\pi R^3$  | Cylinder Volume= $\pi R^2 h$**

This text is displayed as three rows (separated by the vertical-bar | symbol), tiled over the screen. You can set the text to any value and set the color.

## 26.1 ABOUT THE ANTIQUE FONTS

Best calculator comes with three special fonts: a 7-segment display type font which is taken from a

The 7-segment display is from a Litronix 22-segment alphanumeric display. The font that's built into Best Calculator includes all digits, a number of punctuation signs, and the letters A-Z.

The Antique Segment display is taken from US Patent 744923. This display was used for outdoor advertising. They are fully described in the Hawkins Electrical Guide, 1917 editions, in paragraphs 865 to 884.

The Antique Carriage font is perhaps the most interesting of all. It's an early dot-matrix display where the dots are curved segments that are intended to display the digits 0 to 9. They were used in theatres; a person would go to the theatre in their carriage; the carriage would wait until called. The patron would be given "punch card" with their carriage number on it. A mechanical device would take the punch card and display the correct number on the sign, at which point the corresponding carriage would come forward.

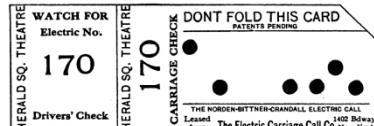
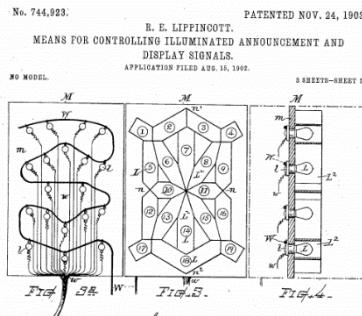


FIG. 219. Punched carriage check for electric carriage call.

For modern eyes, the attempt to make natural digits is wasted; electrically the signs were much more complex than a simple 7-segment display would be, and much less flexible.

## 27 A BRIEF HISTORICAL NOTE ABOUT BASIC

---

As the first programming language designed for students, BASIC holds a special place in the history of programming languages.

```
LET X = (7+8)/3
PRINT X
END
```

*Sample BASIC program from the first Dartmouth BASIC instruction manual, 1964*

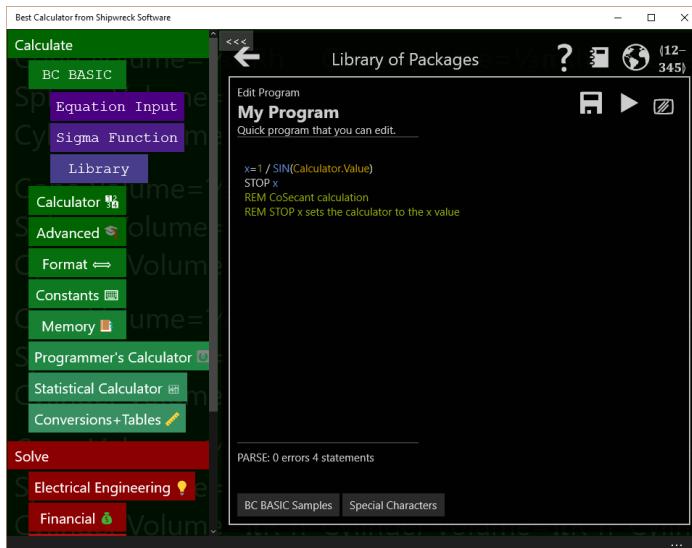
Best Calculator BASIC lets you program your own simple functions into the Best Calculator, extending its abilities to exactly match your needs. Everyone who's a specialist has the same problem with typical calculators: there's some standard calculation in your profession, but no calculator includes those particular functions on the keyboard.

Adding a program to the Best Calculator is easy. The BC BASIC Library key shows you all the programs you've written, neatly organized into individual *packages*. The BC BASIC environment gives you access to a wide variety of sample programs. Naturally, Help is just a click away. Within the programming environment you can add or edit your new program, run your program, and bind your program to any of the five programmable keys on the keyboard.

The programs you write will *roam* between your computers and your phone. You can write a program on one computer, and it will automatically roam to your other computers. (This requires that you've signed in with a Microsoft Account, of course). You can also *Export* your packages to a file, and then *Import* the package into Best Calculator running on another computer.

## 28 EQUATION INPUT: YOUR FIRST PROGRAM

Tap the BC BASIC and then the Equation Input keys. An Edit windows is displayed with a mini program. Your first program is written for you so you can see what a program looks like.



### 28.1 WHAT IS A PROGRAM?

A program is a list of *statements*, each of which performs some action. Many useful programs are just a single equation – for example, to convert feet to acres, or do a financial calculation

### 28.2 HOW DO I RUN MY PROGRAM?

Press the ► (Run Program) key to run the program. When you're in the editor, the final result will be displayed. The final result is either the result of the STOP statement or the result of the last assignment statement.

### 28.3 HOW CAN I PRINT SOME TEXT?

Use the PRINT statement like this:

```
PRINT "Calculation complete!"
```

## 28.4 HOW CAN MY PROGRAM USE THE CALCULATOR VALUE?

The best way is the *Input expression*.

```
X = INPUT DEFAULT 3.4 PROMPT "Enter a tax rate"
```

Input is only for numeric values (you can't get a person's name, for example)

## 28.5 HOW CAN I WRITE AN EQUATION?

Use LET, the *assignment* statement, and an *expression* (equation). For example, suppose you need to calculate the circumference of a circle and the equation is PI times the diameter. Your equation might be

```
LET circumference = PI * diameter
```

Or suppose you need to calculate the inner diameter of a pipe given the outer circumference and the pipe thickness.

```
LET thickness = 3  
LET outerCircumference = 30  
LET outerDiameter = outerCircumference / PI  
LET innerDiameter = outerDiameter - 2 * thickness
```

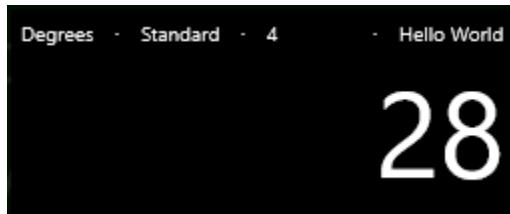
## 28.6 HOW CAN I WRITE TO THE CALCULATOR?

You can write to the calculator in three ways.

You can write a little text message to the top of the calculator display with Calculator.Message.

```
Calculator.Message = "Hello World"
```

Which then show up on the calculator results windows



Use Calculator.Value to get or set the result value



If you end your program with a STOP <value>, the value will be placed into the calculator display. Or, if there isn't a stop, then the last assignment statement (LET statement) evaluated is sent to the calculator. The *equation* or expression form (starting a line with just an equals sign), is considered an assignment even though the value isn't assigned to a variable.

## 28.7 WHY DO SOME EXAMPLES START WITH AN = SIGN?

Lines that just start with an equals sign are the “equation” form. You can put any expression (equation) on the right of the equals, and the value will be computed. If the

## 28.8 HOW CAN I MAKE SEVERAL DIFFERENT PROGRAMS?

See the section on using the library. You can write different programs and give them all different names. They will even roam between your different devices.

## 28.9 WHAT ARE ALL THE KEYS ON THE SCREEN?

The (Back) key will take you to the Library.

The (Help) key pops up this manual

The (Web) key takes you to the Best Calculator web site.

The (Bind key) key lets you program one of the goldenrod (dark yellow) keys with your program. That way you can run your program straight from the regular calculator.

The  (Save) key will save your program.

The  (play) key will run your program. You can also press the F5 key.

The  (test) key will run your programs tests. You can also press the F7 key.

The  (screen) key will clear the colorful output screen. That screen is only displayed if you PRINT output.

Under the edit area is the “PARSE” output; it tells you if your program “parses” correctly – it tells you if and when you make a mistake. Don’t fret too much about making parse errors; experienced programmer make them all the time

The BC BASIC Samples key show you a few simple samples to get started.

The Special Characters key lets you insert some of the hard-to-type characters that BC BASIC can use.

## 28.10 HOW CAN I LEARN MORE?

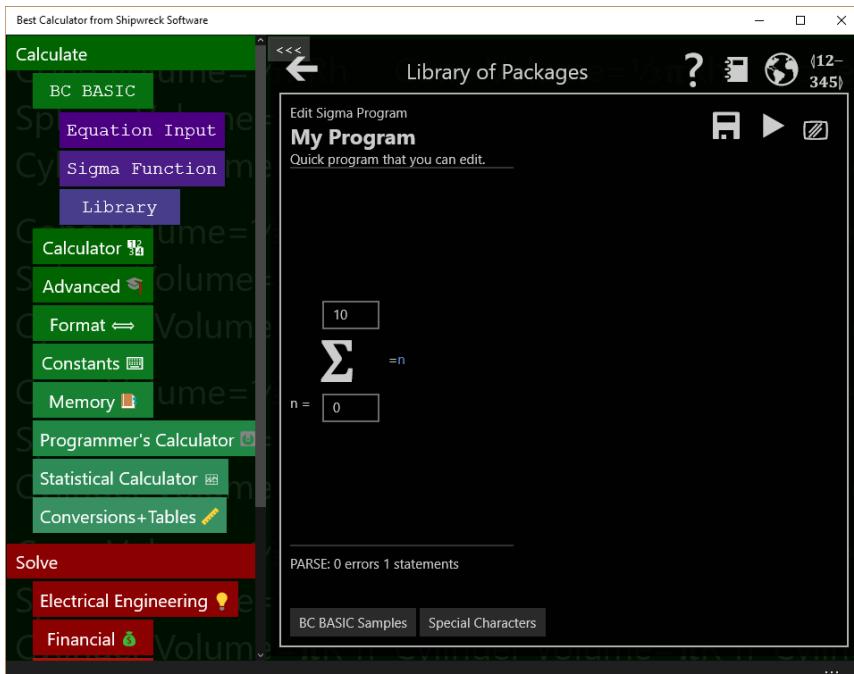
The Reference: Language Basics section tells you everything that makes up a valid BC BASIC program. As a helpful reminder, press the BC BASIC Samples key

to get a little pop-up with some common code snippets. Or press the big  key to see this manual.

A BC BASIC program is a series of statements (lines); each line is an equation. A common statement is *assignment*, which lets you do math. Other common statements are PRINT and INPUT to print results for the user and get numerical values from the user.

## 29 SIGMA FUNCTION: ADVANCED PROGRAMMING

Press the BC BASIC and then the Sigma Function Input keys. An Edit windows is displayed with a mini sigma expression program. Your first expression is written for you so you can see what a program looks like.



The Sigma Function page will run an expression that you provide, summing up the response.



Press ► (Play) or F5 and see the result

The Sigma function is run 11 times. Each time the variable n is set, first to 0, then to 1, and so on up to and including 10. The value of the expression is summed, and the overall value displayed.

Press the →□ key to copy the result into the calculator display.

The rest of the Sigma Function page is the same as the Equation Input page. The Samples include an example of the Taylor expansion to calculate the Sin of a value

```
REM TAYLOR expansion for SIN
REM Convert n(0,1,2,3,4) into series (1,3,5,7,9...)
series = n*2+ 1
x = Calculator.Value
val = x**series / Math.Factorial(series)
isOdd = (Math.Mod (n,2) = 1)
IF (isOdd) THEN val = -val
=val
```

The Taylor expansion for Sin requires a series of numbers 1, 3, 5, 7, and so on. The Sigma function only produces numbers 0, 1, 2, 3, ... .The first step is to convert the n value (0, 1, 2, 3, ...) into a series value (1, 3, 5). This is done with the line series = n\*2+1.

Next the code calculates the  $x^{\text{series}}$  value and divides by  $\text{series}!$ . Odd parts of the sequence are supposed to be subtracted from the sequence, so those are negated.

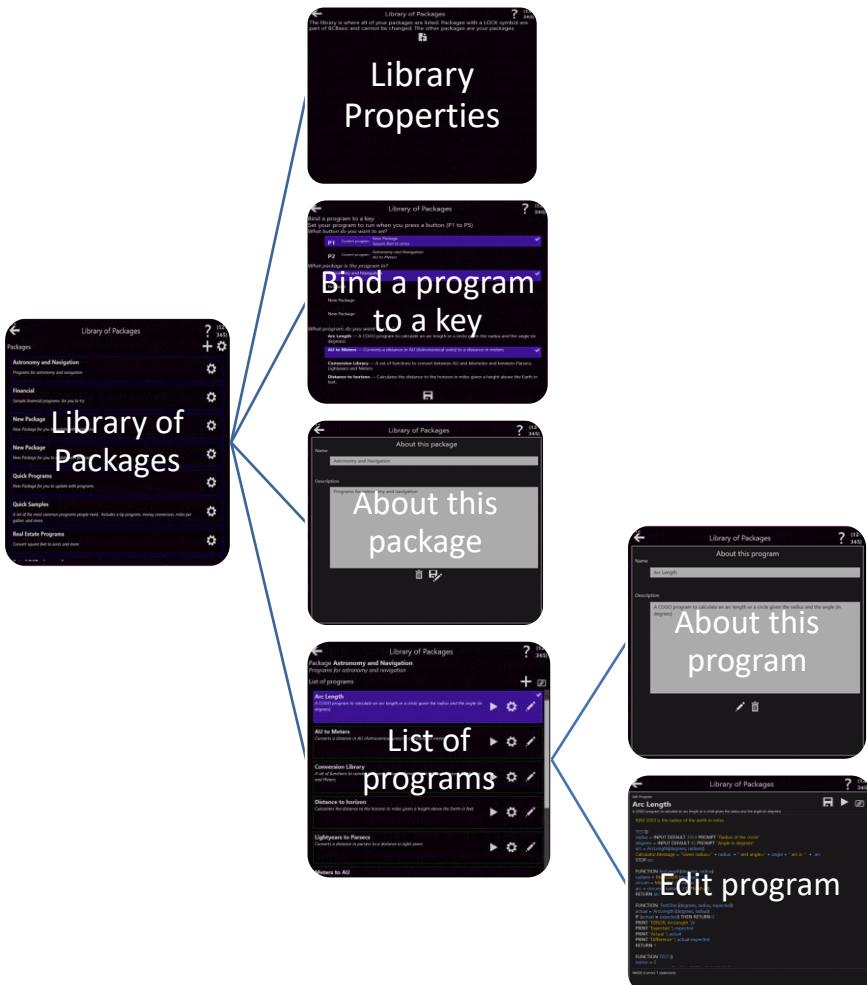
The x value is taken from the **Calculator.Value** number from the calculator display.

Lastly the resulting value for a single series value is return (=val).

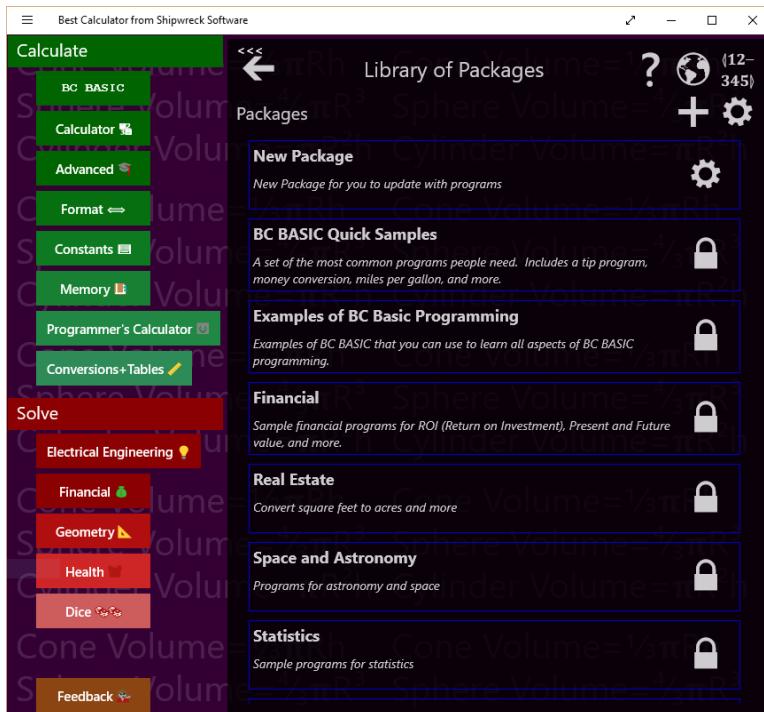
# 30 WHAT ALL CAN YOU DO IN THE BC BASIC ENVIRONMENT?

It's time for a introduction to all of the different dialogs you will use to create and run your programs.

## 30.1 ALL THE MAIN EDIT DIALOGS



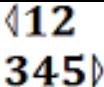
## 30.2 LIBRARY OF PACKAGES



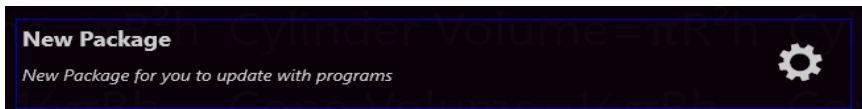
Library of Package is the first library dialog you see. It lists all of the packages that you can run, examine, or change.

In the Library of Packages dialog there are many important controls.

Control	How and when to use it
 Back arrow	<p>The big back arrow is present on all of the dialogs. Press it to return to the parent dialog. If you are in the Library of Packages dialog, the BC BASIC environment will be hidden, and you can see the Best Calculator display.</p> <p>Don't worry! None of your changes are lost. Just press the BC BASIC key to see the Library of Packages dialog again</p> <p>The little "chevron" arrow is a Best Calculator arrow; it hides the Best Calculator menu of calculators.</p>

	Displays the main HELP PDF page using the default PDF reader (often a web browser).
	Goes to Amazon.com. You can buy a copy of the Best Calculator manual from Amazon.
	Goes to the Best Calculator web site
	Displays the Bind a program to a key dialog. This lets you bind one of your programs to one of the P1, P2 and so on key in the Best Calculator keyboard. When you press a key, the program you selected will be run.
	Creates a new package. The new package gets a name which you should change. Use the GEAR icon (  ) to change the name.
	Displays up the Library Properties dialog. From this dialog you can Import a BC BASIC package.

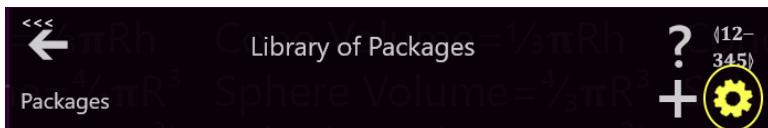
You can also tap on individual packages in the list of packages. Each package entry shows its name and description and includes a GEAR icon to display the package details.



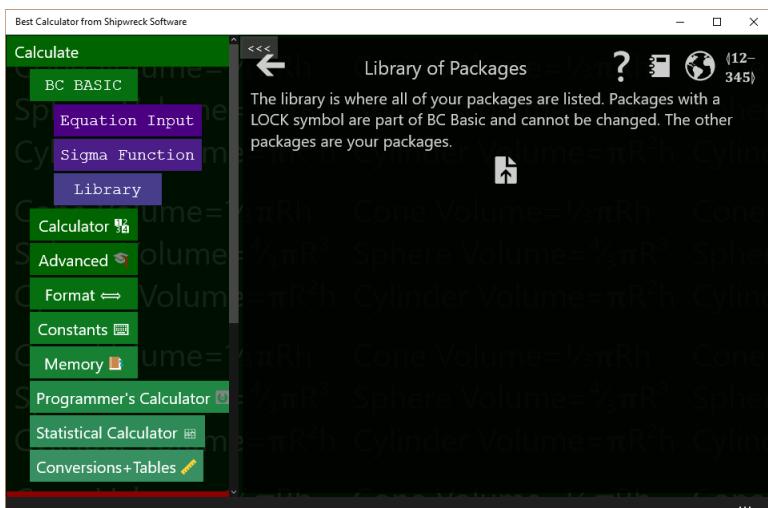
Control	When and how to use it
Tap on a package	Brings up the List of programs for that package. From the list you can add new programs and edit and run your programs.
	Bring up the About this package dialog. From that dialog you can change the name and description for a package.

### 30.3 LIBRARY PROPERTIES (IMPORT BC BASIC PACKAGE)

Bring up the Library Properties dialog by tapping the Library of Packages GEAR



key (  , highlighted). The Library of Packages header lets you go to the help file, bind a program to a key, add a package for your programs or display the library properties screen.



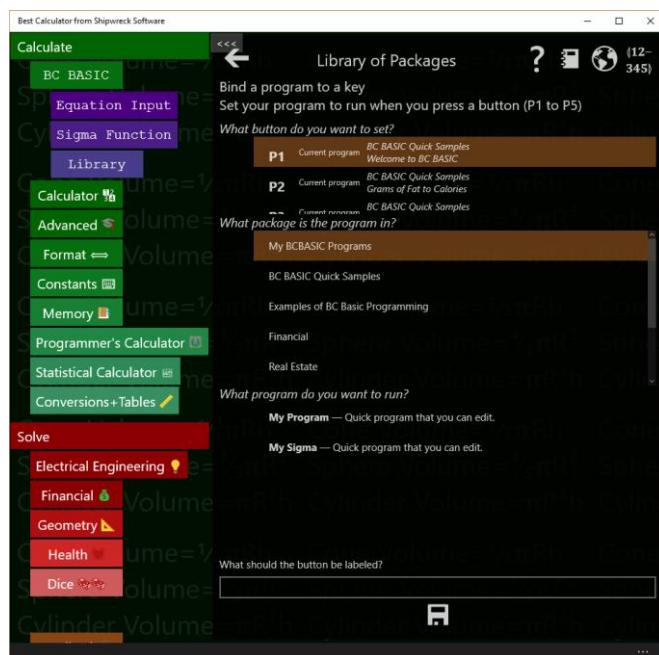
In the Library properties there is a single control

Control	When and how to use it
	Imports new BC BASIC packages from various sources.

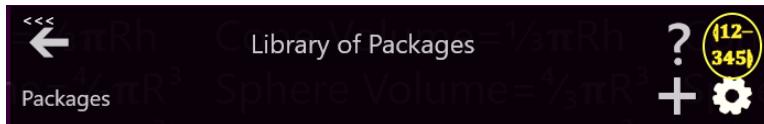
# Guide to Using Best Calculator

## 30.4 BIND A PROGRAM TO A KEY

Page | 81



Bring up the Bind a program to a key screen by tapping the Library of Packages BIND key (highlighted below)



There are several programmable keys which you can bind a program to. When you tap one of those keys, the program that has been bound will be run. When you first get Best Calculator, a selection of programs has already been bound.

To pick a program to run when a programmable key is pressed, select an answer to the three questions in the Bind a program to a key screen and tap SAVE.

The first question is *What key do you want to bind to?* Tap one of the keys in the key list (labeled P1, P2, P3 and so on) to pick a key to bind to. People often just pick key P1. The key list tells you what package and program the key is currently bound to. This helps you pick the right key to use.

The second question is *What package is the program in?* All the possible packages are listed. As you tap on a package, the next list changes to show the programs in that package. Tap on the package with the program you want to run.

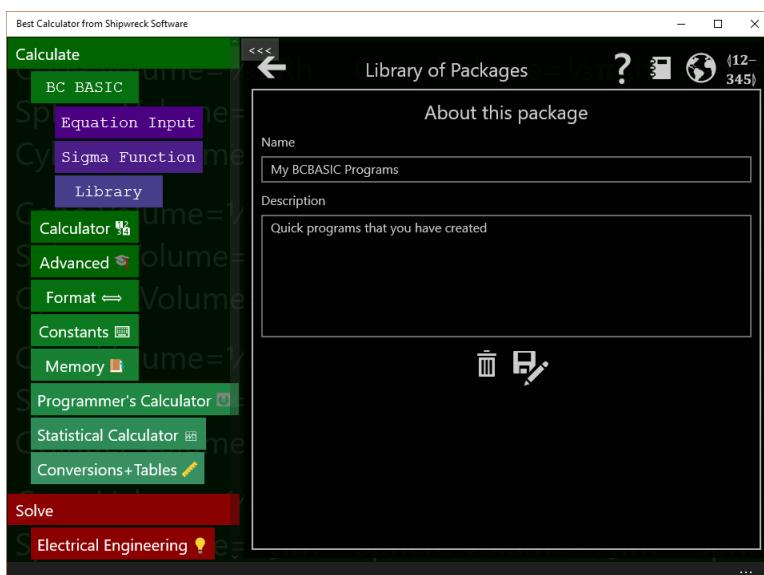
The third question is *What program do you want to run?* Tap the program to run.

Lastly, **be sure to tap the SAVE key (  ).** Your selection isn't saved until you press save.

You can keep on binding programs to more keys, or press the BACK ARROW (  )to go back to your last dialog box.

## 30.5 ABOUT THIS PACKAGE

Bring up the About this package by tapping either the GEAR (  ) or LOCK (  ) in the package listing (see highlighted) in the Library of Packages screen.



The About this package screen lets you change the name and description of a package, delete it, or export it (save it to an external file).

You can only modify your own packages (the ones with a GEAR icon). Packages that came with Best Calculator (with a LOCK icon) cannot be modified. You can copy a locked package to your own library and can edit it there.

### BC BASIC Quick Samples

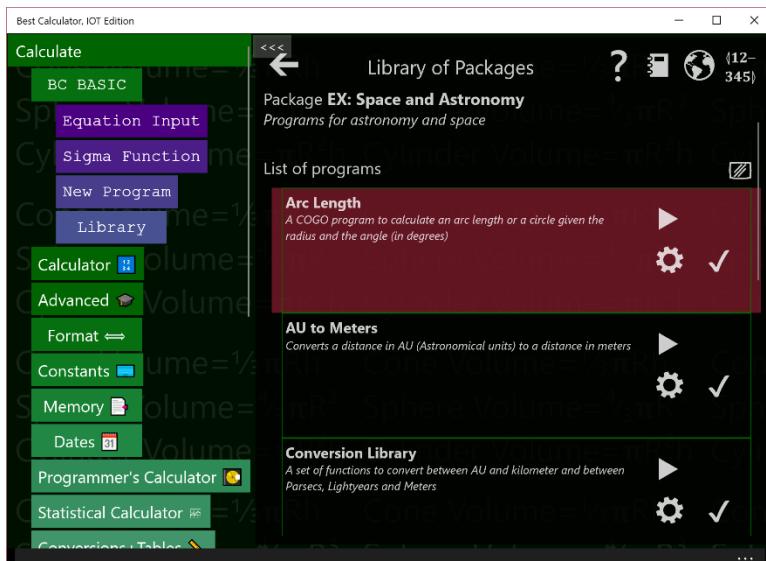
*A set of the most common programs people need. Includes a tip program, money conversion, miles per gallon, and more.*



Control	When and how to use it
Name	You can change the name of the package here. Just enter a new name. The name is automatically saved.
Description	You can change the description of the package here. Just enter or modify the description. It will be automatically saved.
	Deletes the package. Once deleted, you will not be able to bring the package back. You will be prompted to confirm the delete.
	Saves (exports) the package (and all the programs in it). You will be prompted for a file name to save to.  Once you Save (Export) a package, you can store it in OneDrive, save to a web page, or email to a friend or coworker. They can Import the package from the Library Properties page.

## 30.6 LIST OF PROGRAMS

Bring up the list of programs by tapping on a package in the Library of Packages screen.

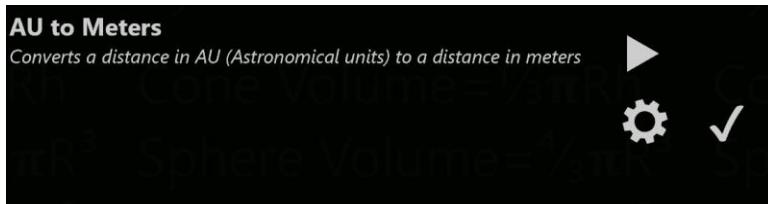


Packages contain multiple programs which you can run. The List of programs screen lets you see all and run the programs in a package, add new programs, edit programs, and more.

Control	When and how to use it
Add Program	Adds a new program to the package. Once created, you will need to tap the GEAR (  ) to set the program's name and description.  You can only add programs to your own packages. Packages that came with Best Calculator cannot be modified.
Clear Screen	Clears the output screen. If the output screen is not visible, you won't see a change (but it will be cleared nonetheless)

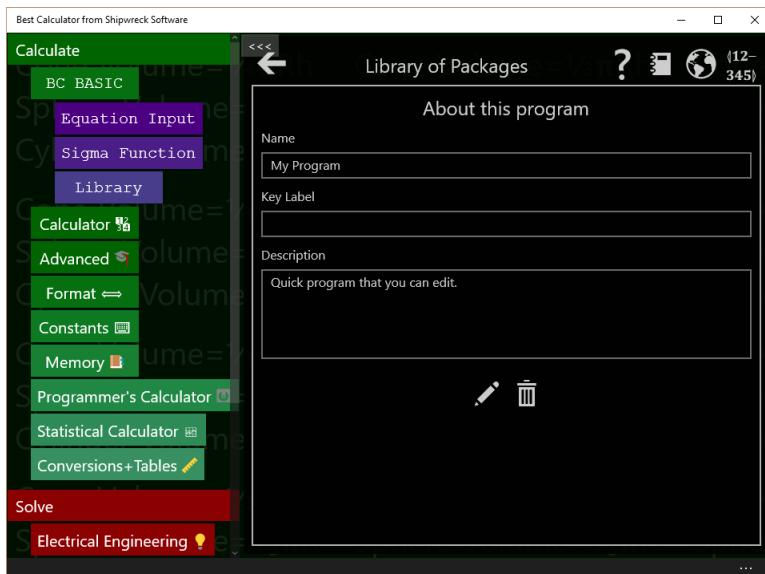
You can also tap on individual programs in the list of programs. Each program entry shows the name and description of the program and lets you run and

examine the properties of the program. If the program is one you wrote, you can also edit it.



Control	When and how to use it
Tap on a program	Does nothing in particular ☺
	Runs the program from the start
	Runs all of the tests in the program and at the end gives you a summary of the test results.
	Displays the About this program dialog. From that dialog you can edit the program name and description.
	<p>Display the Edit Program dialog. From that dialog you can edit and run the program.</p> <p>Packages that come with Best Calculator cannot be modified. For those programs, the Edit key will show you the source code for the program but will not let you change it.</p>

## 30.7 ABOUT THIS PROGRAM



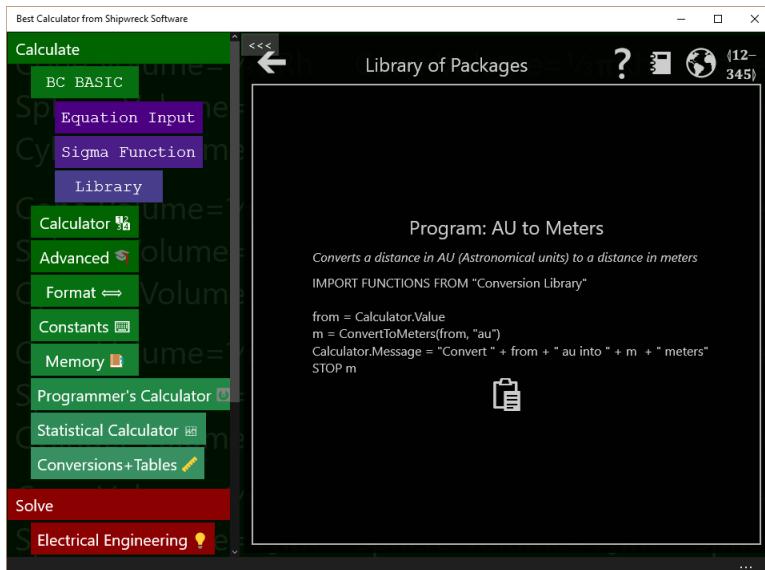
Bring up the About this program screen by clicking the GEAR icon in the program entry in the List of programs (see highlighted). The About this program screen lets you modify the name and description of a program, go straight to the edit program screen, or delete the program.



Control	When and how to use it
Name	You can change the name of the program here. Just enter a new name. The name is automatically saved when the program is saved.
Description	You can change the description of the program here. Just enter or modify the description. It will be automatically saved. When the program is saved.
	Display the Edit Program dialog. From that dialog you can edit and run the program.

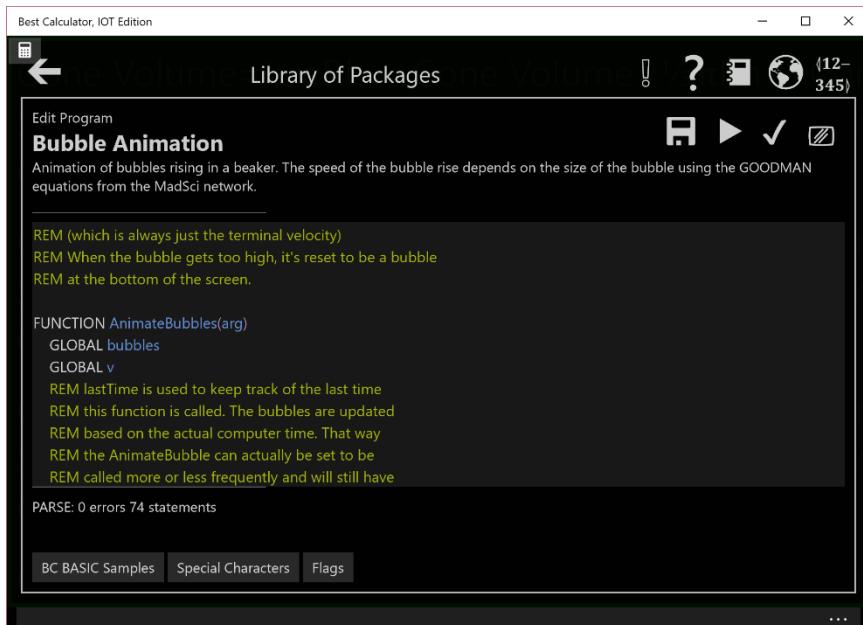
Edit program	You cannot edit programs that come with Best Calculator
 Delete	Deletes the program. Once deleted, you will not be able to bring the program back. You will be prompted to confirm the delete.

Programs that come with Best Calculator are locked and cannot be changed. Their About screen is a little different.



Control	When and how to use it
 Copy to clipboard	Copies the source code for the program to the clipboard.

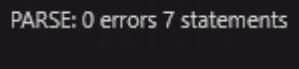
## 30.8 EDIT PROGRAM



The Edit program dialog is where you enter your BC BASIC code.

Bring up the Edit program dialog by tapping the EDIT key on either the program list or by tapping the EDIT (  ) key in the About this program screen.

Control	How and when to use it
	Saves your changes. Your changes are automatically saved when you press RUN. Your program is saved as part of the package file; this is managed for you.
	Runs the program from the start. Some common errors when you press RUN but your program does not appear to run are: <ol style="list-style-type: none"> <li>1. Your program might have a syntax or other error that prevents it from running. You will know because the parse indicator will show an error</li> </ol>

	2. Your program might have run but didn't display anything to the screen. The output is often displayed on the calculator display. Press the Calculator key to see the calculator screen.
 Test Program	Runs the tests for the program. All functions that start with TEST will be run, and the results from each ASSERT are the results. After the tests run the results will be summarized.
 Clear screen	Clears the output screen. If the output screen is not visible, you won't see a change (but it will be cleared nonetheless)
 PARSE: 0 errors 7 statements Parse indicator	This indicates if your program compiles. BC BASIC automatically compiles your program as you type it, and tells you of any syntax errors.  The editor uses <i>syntax coloring</i> on your code; different parts of the code will be displayed in different colors. When a syntax error is discovered, only the program up to the syntax error is colored; the rest shows up in white.

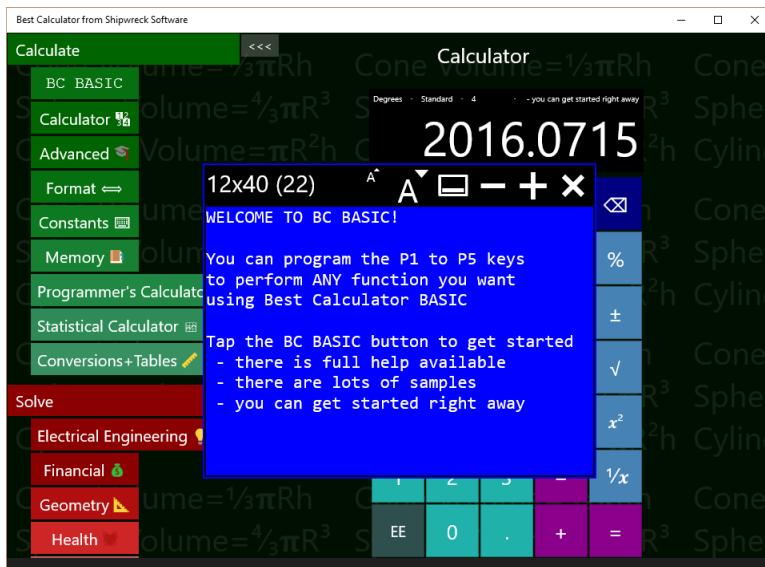
The BC BASIC editor is a stripped-down edit experience; it doesn't have a lot of commands to get in the way of editing your program. The edit commands available are

Key	Command
^F Find	Lets you search for text in your program
^G Go to	Goes to a specified line in your program
^[ Indent	Indent the line or lines that you've selected
^] un-indent	Un-indsents the line or lines that you've selected

There are three special keyboard keys while you are editing a program

Key	How and when to use it
Escape	Stops the program that's currently running. This is useful when you've written an infinite loop (a program that doesn't end)
F5	Acts like the Run program button. Press F5 to start the program running.
F7	Acts like the Test program button. Press F7 to start the tests.

## 30.9 OUTPUT SCREEN



The output screen is where the results of the PRINT, CONSOLE and DUMP commands are written. It is the screen that's cleared or whose color changes when CLS or PAPER is run. You can only write to the screen; you can't read back from it.

The output screen contains both the fixed-character size screen (which is the normal output screen) and the scrolling console output. The console output is mostly intended for debugging.

The controls at the top of the screen let you modify the screen's looks.

Control	How and when to use it
	The output screen always tells you how large the screen is and the font size. Screen sizes include 12x20, 12x40, 16x60 and 24x80
	Reduces the font size, automatically making the output screen smaller.
	Increases the font size, automatically making the output screen larger
	Toggles the console portion of the output screen on and off. Unlike the output screen, the console is a long scrolling list of output.
	Reduces the number of characters on the screen.
	Increases the number of characters on the screen.
	Closes the screen. The screen contents are not lost; when the screen is re-shown, the original contents will still be present. When you PRINT to the screen, it will be displayed automatically.

### 30.10 RUNNING ✓ TESTS IN BC BASIC

By writing a good set of tests, you can be more confident that your program works both when you first write it and later when you update it. BC BASIC has a simple system for running tests: you simply write functions in your programs whose name starts out as TEST, and when you tap on the Run Test (✓) button, the tests will be automatically run.

Here is the test function for the Arc Length program (part of the Space and Astronomy package). It verifies that the function works for a variety of inputs where the results are already known.

```
FUNCTION TEST ()  
    ASSERT (ArcLength (3959, 45) ≈ 3109.391)  
    ASSERT (ArcLength (10, 90) ≈ 15.707963)  
    ASSERT (ArcLength (0, 90) = 0)  
    ASSERT (ArcLength (10, 0) = 0)  
RETURN
```

The simplest way to write a test is to make a function whose name starts with TEST and uses the ASSERT function. The ASSERT function takes an equality expression (an expression that uses any of  $<$   $\leq$   $=$   $\geq$   $>$   $\neq$   $\cong$  or  $\not\equiv$ ) The tests will finish and will tell you which tests passed and which failed.

The Run Test (✓) button is available on each program, in the settings for each package, and in the overall Library setting. If you're editing a program, you can also just press the F7 key to run the test for just that program. Each one will run more tests: the one on each program will run only the tests in that program; the one in the Package settings will run all of the test in all of the programs in the one package. The one in the Library setting will run all the tests in all of the packages and programs.

A different way to write a TEST function without using ASSERT is to return the number of errors that the test found. If it doesn't return a number that is zero (all the tests passed) or more than zero (the number of failures), then that is an error, too.

## 31 BASIC LANGUAGE REFERENCE

---

### 31.1 PROGRAM STRUCTURE

A BC BASIC program is a list of *statements* and *functions*. Each statement can optionally start with a *line number*; line numbers are simple integers. Line numbers do not have to be in any special order (this is unlike many version of BASIC where the lines must be in numerical order).

#### Examples of statements

```
FOR I = 1 TO 4
PRINT "Hello World"
NEXT I
100 REM this statement has a line number
```

Statements are normally exactly one line. They can be extended with visible “return” type characters:

↓	U+21B2	DOWNWARDS ARROW WITH TIP LEFTWARDS
↙	U+21B5	DOWNWARDS ARROW WITH CORNER LEFTWARDS
↘	U+2936	ARROW POINTING DOWNWARDS THEN CURVING LEFTWARDS

Statements do not have a statement terminator (for example, the “C” language terminates statements with semicolons). Statements start with a command name like PRINT except that the CALL and LET command names are optional. Examples of statements include **CLS BLUE** and **LET a=10**. The CALL and LET commands are optional; **LET a=10** is the same as plain **a=10**, and **CALL PrintName()** is the same as **PrintName()**.

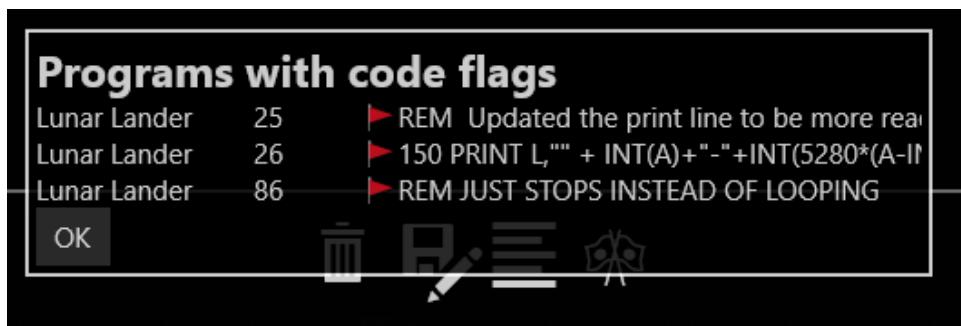
Lines are technically a sequence of characters that ends with a \n, \r or \v. The \v (vertical tab) is sometimes used by Microsoft Word when cut-and-pasting text.

## 31.2 FLAGS

BC BASIC lets you include flags (like and ) in your source code. Adding flags let you mark different lines of your program, either for you to find again later or to let other people review your code. The BC BASIC editor includes a **Flags** button that lets you easily insert a variety of flags into your code.

Comments start with the REM command; the rest of the line is the comment.

The properties page for each package includes a button; press it and each line of each program in the package that includes a flag character will be displayed.



## 31.3 NUMBERS AND STRINGS AND VARIABLES

Numbers and strings are the two most common things people want to manipulate. Inside a BC BASIC program, you can write number constants (like, 4 or 3.14) and string constants (like, “apple”).

### 31.3.1 Numbers

Number can be any of

- Integers like 3 or -5
- Integers using hexadecimal (base 16) notation like 0x65
- Floats (which are stored as double-precision) like 1.2, or -7.8
- Numbers in exponential notation like 45E3 (which is equal to 45000)
- The  $\infty$  symbol is infinity

Doubles which are smaller than 1 can start with just a decimal place (0.3 and .3 are both OK). To help improve the look of your programs, BC BASIC allows for

either a computer-style minus or a wider Unicode minus and dash characters. Example: computer-style is - and wider Unicode is – or –. Microsoft's Word program sometimes converts one into the other.

### Example of using different types of minus signs:

```
REM Constant numbers
```

```
V1 = -12.34E-6
```

```
V2 = -12.34E-6
```

```
V3 = -12.34E-6
```

```
REM Expressions
```

```
E1 = V1 - V2
```

```
E2 = V1 – V2
```

```
E3 = V1 – V2
```

```
REM Negated Values
```

```
N1 = - E1
```

```
N2 = - E2
```

```
N3 = - E3
```

Technically, these are Unicode characters HYPHEN MINUS (U+2D), EN DASH (U+2013) and MINUS SIGN (U+2013). The special Unicode plus signs are not accepted or the other Unicode minus signs.

#### 31.3.2 Strings

String constants can use either computer-style quotes like "" or “smart” quotes. A string that starts with an opening smart must be ended with a smart end quote.

You can't nest smart quotes: the string “hello “special” world” is incorrect. The string will be ended at the end of the word special; the rest of what you think is the string will be misinterpreted. The correct way to write that string is to double-up the ending smart quote like this: “**hello “special”” world**”

You can escape double-quotes by using two of them OR by replace the quote with &QUOT;. In addition, because Word converts two double-quotes into a "" quoted-pair, a quoted-pair will be replaced with a forward smart-quote.

Examples:

```
| PRINT “This is “quoted””.”
```

```
PRINT "This is &QUOT;quoted&QUOT"  
PRINT "This is ""quoted"""
```

BC BASIC includes functions for manipulating strings. You can concatenate strings together with the + operator, get the length of a string with the LEN function, and extract parts of a string with the LEFT, MID and RIGHT functions.

### 31.3.3 Variables, GLOBAL, and DIM

BC BASIC allows you to create and use variables at any time. Variables are given names; names must start with a letter (a to z and A to Z) and from then on can contain letters, numbers, and underscores. Variables can optionally end with a dollar sign (\$). Variables are case sensitive; the variable “name” is different from the variable “NAME”.

Starting in version 3.18, variables can also start with Greek and Coptic letters and can include (but not start) with subscript letters. For example, when calculating how to calibrate a sundial, you might need to calculate the sun’s declination. The declination is often described in textbooks using a delta, and so should our code:

```
FUNCTION SunDeclination(dayOfYear)  
    δ = 23.45 * SIN ( 2 * PI * ((dayOfYear + 284) / 36.25) )  
RETURN δ
```

Similarly, you might see local latitude being written using a phi ( $\phi$ ) and might want your code do match your textbooks like this:

```
FUNCTION SunAzimuth (δ, φ)  
    cosa = -SIN(δ)/COS(φ)  
    A = ACOS (cosA)  
RETURN A
```

Arrays can have named properties for the individual values. This is done using the array SetProperty (name, value) method.

In BC BASIC the “\$” at the end of a variable has no special meaning. The \$ is allowed for compatibility with other versions of BASIC. In some versions of BASIC, a “\$” indicates that a variable is a string. In BC BASIC, any variable can be any kind of variable.

Variables created inside of functions are *scoped* to the function; they cannot be used out of the function.

Use the DIM statement to make array variables. Array variables let you get and set a lot of values in one variable. They are often used in mathematical analysis. See documentation for DIM for full information on what you can do with arrays.

Use the GLOBAL statement in a function to use variable declared outside of the function. Normally you can use the same variable name both globally and in a function and they will refer to different variables. Sometimes in a function you need to refer to a global variable (for example, this is common in callback functions). In these cases, use **GLOBAL variable**. After that, references to the variable will be to the global variable.

### Example of using variables:

```
LET a = 10
LET b = a + 3
LET c = 7.89 / b
LET bignum = -1.3E23
LET smallnum = 2.78E-23

LET name = "Person of interest"
LET information$ = "You can use smart quotes"

LET dog_name = "Sumi"
LET check9 = 99

CLS GREEN
PRINT "Sample variables"
DUMP
```

Each variable includes some properties that tell you about the variable.

**variable.Type** says what the type of the variable is. Possible results include NUMBER, STRING, ARRAY, OBJECT, ERROR, and NOTHING. Other objects (like the different Bluetooth objects) may return other values.

**variable.IsNumber**, **variable.IsString**, **variable.IsObject** and **variable.IsError** are true if the variable is a number, string, object or error respectively.

**variable.IsNaN** is true if the variable is number which is a NaN (not a number) value. If the number is a string or something that isn't a number, will return true. The Math extension also includes a Math.IsNaN function.

When variable.IsError is true, you can get the ErrorCode and ErrorString properties from the error object. These can be used to decide how to handle the error.

```
IF (value.IsError)
    PRINT value.ErrorCode
    PRINT value.ErrorString
END IF
```

## 31.4 <EXPRESSION> OVERVIEW

### 31.4.1 Quick introduction to expressions

“1 + 1” is one of the simplest expressions; it adds two *constants* (the two 1 values). Best Calculator BASIC has the normal set of operators and precedence rules for modern computer languages, plus a few extra convenience functions.

### 31.4.2 Expression Rules

BC BASIC is designed to make most expressions work normally.  $1+2*3$ , for example, will multiple the  $2*3$  first, and then add the 1. You can put parenthesis around your expressions. You can use either curved parenthesis or square brackets.

Expression type	Explanation and Sample
Variable	When evaluated, is the value of the variable. Examples: A B\$
Constant	A numeric or string constant Examples: 1 3.14 0xFF “Hello World” “She said, &QUOT;Hello&QUOT;”
Named values PI and RND	PI is always equal to PI (3.1415...). It’s more common in BC BASIC to use the Math.PI value and not PI by itself. RND is a new random number between 0 and 1.

( expression ) [ expression ]	You can place expression inside of parenthesis () or square braces [] to show which operations should happen first.
expression OPERATOR expression	Any of the standard math operators like + and -. See the table below for a full list. BC BASIC also includes a variety of comparison and logical operators, and the INPUT operator.
PREFIX expression expression POSTFIX	Use minus (-) to negate a number. Use power and root operators to take a square root, or raise a value to a power.
Function ( argument, argument )	The value of the given function. There are a set of built-in functions (SGN ABS SIN COS TAN ASN ACS ATN LN EXP SQR INT LEN CODE CHR\$), or you can define your own. The parentheses are normally required. Unlike some other versions of BASIC, you cannot omit the parenthesis for the built-in functions.

Constants are always handled as doubles (1.2, or 0.1, or -56.7, or just plain 4 or -2

Example of expressions are

Example	Explanation
1	A constant
apple	The value of the variable "apple". The variable should have already been defined; otherwise it's assumed to be a "NaN" (Not a Number)
SIN (PI / 4)	The SIN of $\frac{1}{4}$ PI radians. The trigonometry functions all take their values in radians. Use the Math.DtoR() function to convert degrees to radians.
1 + 2 * 3	Is the value 7 (and not 9); multiplication is higher priority than addition so the $2*3$ is done first and then the 1 is added to it.
(1+2) * 3	Is the value 9; the parenthesis force the addition to be done first
"A" < "B"	Is the value 1 (for 'TRUE') because A is sorted before B.

### 31.4.3 negate, power, root prefix and postfix operators

Operator	Explanation and Sample
$2^3 4$	<p>Square, cube and fourth power operators. BC BASIC is special among most programming languages for allowing these superscripts to be used. You can also use the Math.Pow() function or the <math>^{**}</math> operator</p> <p>Example:  <math>5^2</math> is 25 because <math>5*5</math> is 25  <math>\text{Math.Pow}(5, 2)</math> is also 25  <math>5^{**2}</math> is also 25</p>
$\sqrt[3]{\sqrt[4]{}}$	<p>Square root, cube root and fourth root operators. You can also use the Math.Pow () function or the <math>^{**}</math> operator</p> <p>Example:  <math>\sqrt[3]{64}</math> is 8 because <math>8*8</math> is 64  <math>\text{Math.Pow}(64, 1/3)</math> is also 8  <math>64^{**0.3333333333333333}</math> is also 8</p>
-	<p>Unary minus converts any number to its negative</p> <p>Example:  <math>12 + -3</math> is 9 because <math>12 - 3</math> is 9</p>

### 31.4.4 Operators + - \* / and more

Each operator has a precedence value; operators with a higher precedence will be done before operators with a lower precedence.

Operator	Precedence	Explanation and Sample
$^{**}$	10	<p><math>^{**}</math> is the “raise to the power” operator</p> <p>Example:  <math>2 ^{**} 6</math> is 64 because <math>2*2*2*2*2*2</math> is 64</p> <p>Historical note: early versions of BC Basic includes a “root finding” operator. This proved to be confusing in practice, and has been removed.</p>
$* /$	9	$*$ is the computer sign for multiply

		/ is the standard computer sign for divide Examples: 3 * 4 is 12 10 / 2 is 5
+ -	6	<p>+ is the standard computer sign for addition or string concatenation - is the standard computer sign for subtraction</p> <p>Example: 1+2 is 3 10 - 2 is 8 "1" + 2 is "12", but 1 + "2" is 12. When the left side of a + is a string, both sides are treated as string and concatenated together; when the left side is a number, both sides are treated as numbers. Strings that cannot be converted are handled as a NaN</p> <p>Extra bonus: The Unicode character set designates three characters that are commonly used for minus signs:</p> <ul style="list-style-type: none"> <li>HYPHEN-MINUS (U+2D), the normal minus sign</li> <li>MINUS SIGN (U+2212)</li> <li>EN DASH (U+2013)</li> </ul> <p>Any of these can be used for a minus sign. Among other things, this makes it easier to copy your programs back and forth between BC BASIC and Microsoft Word.</p>
< <= = > = > <> ≈ ≠	5	<p>The normal set of operators for less than, less-than-or-equal, and so on. The &lt;&gt; operator is for "not equals".</p> <p>The ≈ is for "approximately equals"; for numbers it means that the two numbers are within 5 significant figures of each others, and for strings that they compare</p>

		<p>equal using the CurrentCultureIgnoreCase flag.</p> <p>Note that two numbers which are different signs are never considered approximately equal even if they are both really, really close to zero.</p> <p>The <math>\not\approx</math> symbol is “Not approximately equal to” and is the opposite of <math>\approx</math>.</p>
NOT	4	<p>Inverts the logical value of its argument.</p> <p>Example</p> <p>IF NOT A=B THEN &lt;statement&gt; will do the statement if A is NOT equal to B.</p> <p><b>Note on logical values:</b></p> <p>The logical operators (NOT, AND, and OR) can take any numerical value and treat it as a logical value; anything that's zero is treated as FALSE and all other numeric values are TRUE. The operators will only ever produce a 0 or 1.</p>
AND	3	A logical AND operation; A AND B will be 1 if both A and B are TRUE (non-zero).
OR	2	A logical OR operation. A OR B will be 1 if either A or B are TRUE (or if both of them are).
INPUT		<p>INPUT is a complex operator with two optional values, a PROMPT and a DEFAULT value. When your program runs, a dialog box will pop up with the prompt you specific and with a starting value of whatever the default value was (it can be an expression). The user then enters a value and presses the OK key; the resulting value is the value of the INPUT expression.</p> <p>Example:</p> <p><code>birth_year = INPUT DEFAULT 1967 PROMPT "Enter your birth year:"</code></p>

**Examples of expressions:**

```
REM Multiplication binds more than addition  
REM a will be 7 (1 + (2*3)) and not 9 ((1+2) * 3)  
a = 1 + 2 * 3  
b = 10 - 4 / 2
```

```
REM Demonstrate cube root and raise to 4th power  
c = 3 ∛ 10  
d = 6 ** 4
```

```
REM Comparing values. PI is not about 22/7  
REM but it is about 355/113  
e = PI ≈ 22/7  
f = PI ≈ 355/113
```

```
REM These are all false (except j)  
g = PI > 22.7  
h = PI >= 22.7  
i = PI = 22.7  
j = PI <> 22.7
```

```
REM Logical operators  
k = c > 2 AND c < 4  
l = c > 2 OR d < 10  
m = NOT (c > 2 OR d < 10)
```

```
REM You can ask for input from the user  
n = INPUT DEFAULT 5 PROMPT ↓  
    "Please enter a number"
```

```
CLS BLUE  
PRINT "All the variables"  
DUMP
```

**Example of using the RND and PAUSE statement to make a little random animated display:**

```
CLS BLUE  
COUNT = 0  
  
REM You can also use a FOR..NEXT loop  
10 A = DrawDot()
```

```
COUNT = COUNT + 1
IF (COUNT > 100) THEN GOTO 20
PAUSE 1
GOTO 10
20 PRINT AT 1,1 "DONE"

FUNCTION DrawDot()
col = INT (RND * Screen.W) + 1
row = INT (RND * Screen.H) + 1
ch = "*"
PRINT AT row,col ch
RETURN
```

### 31.4.5 The INKEY\$ expression

The INKEY\$ expression returns the most recent key that the user pressed on the keyboard. This is often used in games because the user doesn't have to enter a value into a text box. If no key has been pressed, INKEY\$ returns "".

Here's a little sample "game" where the user can press the WASD keys to move a balloon around on the screen. A graphics screen (g) is created and a small balloon is made using a Text element. Then the INKEY\$ value is examined in a loop; each of the different WASD characters moves the balloon's X1 or Y1 position. Then the whole thing is done again.

```
CLS "#3377BB"
PRINT "BALLOON GAME"
g = Screen.Graphics (50, 50, 200, 200)
g.Border = "#3377BB"
balloon = g.Text (0, 0, 40, 60, "🎈", 40)

10 REM Top
PAUSE 10
cmd = INKEY$

xd = 0
yd = 0
REM Figure out how we want to move.
IF (cmd = "X") THEN STOP
IF (cmd = "W") THEN yd = 3
IF (cmd = "A") THEN xd = -3
IF (cmd = "S") THEN yd = -3
IF (cmd = "D") THEN xd = 3
balloon.X1 = balloon.X1 + xd
balloon.Y1 = balloon.Y1 + yd
GOTO 10
```

Technically, all the keys that are pressed and which don't go into a text box are placed in a queue (first-in, first-out); INKEY\$ returns the oldest keypress which hasn't been returned yet.

### 31.4.6 The INPUT expression

The INPUT expression lets you prompt your user to enter a value. You can use DEFAULT <value> to supply a default value and a PROMPT <string> to specify a prompt string. You must specify the DEFAULT before the PROMPT.

If the DEFAULT value is a string, the user will be allowed to enter either a number or a string. Otherwise, the user may only enter a number. The original version of BCBASIC only allowed for numeric entry.

There is also an INPUT statement that is less powerful than the INPUT expression. It's provided for compatibility with other versions of BASIC.

#### Example of the INPUT expression:

```
LET interest = INPUT DEFAULT 3.5 ↴  
PROMPT "Interest rate"
```

The user will enter a value which will be interpreted as a number.

## 31.5 MATH FUNCTIONS

BC BASIC includes a small but powerful set of math functions. These are designed to be compatible with other versions of BASIC.

Some version of BASIC let you use these functions without parenthesis; BC BASIC does not.

There are even more functions available in the Math extension.

### 31.5.1 Trigonometry functions SIN COS TAN ASN ACS ATN

Function	Notes
ACS(value)	Calculates the inverse of COS; given a value will compute the corresponding angle in radians.

ASN(value)	Calculates the inverse of SIN; given a value will compute the corresponding angle in radians.
ATN(value)	Calculates the inverse of TAN; given a value will compute the corresponding angle in radians. Note that TAN of 90° is infinite.
COS (radians)	Calculates the cosine of an angle given in radians
SIN (radians)	Calculates the sin of an angle given in radians
TAN (radians)	Calculates the tangent of an angle given radians

### 31.5.2 Logarithm and Power functions LN EXP SQR

Function	Notes
EXP (value)	Calculates the value $e^{\text{value}}$ for any given value. This is the inverse of the LN function This is the same as Math.Exp(value)
LN (value)	Calculates the natural (base e) logarithm of the given value. This is the inverse of LN. This is the same as Math.Log(value)
SQR (value)	Calculates the square root of a value. You can also use the √ square root operator, or use the ** power operator. This is the same as Math.Sqrt(value).

### 31.5.3 Rounding and sign functions SGN ABS INT

Function	Notes
ABS (value)	Calculate the absolute value of a number. This is the same as Math.Abs(value)
INT (value)	Calculates the floor of a number. The floor is the number rounded down to the nearest integer. For example, INT (12.8) is 12 and INT (-12.8) is -13. This is the same as Math.Floor (value)
SGN (value)	Return the sign of a number. The sign is 1 for positive values, -1 for negative values, and 0 for zero. This is the same as Math.Sign(value)

## 31.6 STRING FUNCTIONS (LEFT, MID, RIGHT, LEN, CHR\$, CODE, SPC, VAL)

### 31.6.1 LEFT (string, count), MID (string, index, count) and RIGHT (string, count)

The LEFT, MID and RIGHT functions get data from the start (LEFT), middle (MID) or end (RIGHT) ends of a string. The return value is always a string.

For the MID function, you don't need to provide the count value; it will be assumed to be the rest of the string. The index for the MID function starts at 1; **MID (string, 1, count)** is exactly the same as **LEFT (string, count)**.

#### Examples of the string functions:

```
REM LEFT(string, count) returns string 'count' long
REM from the left part of the input string
REM The example will print AB
PRINT "LEFT("ABCDE", 2) = "; LEFT("ABCDE", 2)
```

```
REM MID(string, index, count) returns
REM      a string 'count' long
REM from the middle part of the input string
REM starting at 'index'. The first letter is index 1.
REM The example will print BCD
PRINT "MID("ABCDE", 2, 3) = "; MID("ABCDE", 2, 3)
```

```
REM RIGHT(string, count) returns
REM      a string 'count' long
REM from the right part of the input string
REM The example will print DE
PRINT "RIGHT("ABCDE", 2) = "; RIGHT("ABCDE", 2)
```

Each of the functions has similar special cases.

1. The count is always truncated so that the return value doesn't go past the size of the string
2. If the count value is less than one, then a blank (zero length) string is returned

3. If the index for MID is more than the length of the string, then a blank string is returned

### 31.6.2 LEN string

The LEN function (which has optional parenthesis) returns the length of a string. Blank strings have a length of zero.

#### Examples of the LEN function:

```
REM The length is always 3
PRINT "LEN of string ABC is 3"
PRINT LEN "ABC"
PRINT LEN ("ABC")

REM Length of a blank string is zero
PRINT LEN ""

REM LEN PI is the length of the string
REM      representing the number PI
REM (the answer is 16 printed digits)
PRINT LEN Math.PI
```

Special cases include:

1. When given a number, returns the length of the string that represents the number
2. Blank strings have zero length
3. Unicode strings that use *surrogate pairs* will count each surrogate pair as two characters. For example, the Unicode U+1F60B character point (FACE SAVORING DELICIOUS FOOD) is presented as two characters, U+D83D and U+DE0B. These two characters are called a surrogate pair, and represent the Unicode character.

### 31.6.3 CHR, CODE (and ASC)

CHR converts a series of numerical Unicode values into a string. There is also, for compatibility, a CHR\$() that takes a single argument. The CHR\$ function is technically an expression operator and does not require parenthesis. That is, CHR\$ (65) can be written as CHR\$ 65.

The CODE function returns the Unicode code point of the first character in a string. The CODE function is technically an expression operator and does not require parenthesis. That is, **CODE ("A")** can be written as **CODE "A"**. The ASC function is exactly the same as CODE; it's included for compatibility with other versions of BASIC.

In Windows, Unicode code points outside the Basic Multilingual Plane are encoded as surrogate pairs and are handled as two characters.

#### Example of the CHR, CHR\$ and CODE functions:

```
REM CHR converts a Unicode character number
REM      to a string
REM 65 is the ASCII A
REM Unicode U+1F60B is
REM      "FACE SAVOURING DELICIOUS FOOD".
REM It is converted into two chars
REM      (a surrogate pair).
REM CHR$ is the same function, but takes in
REM      only one parameter
PRINT "CHR (65) = "; CHR (65)
PRINT "CHR Unicode: "; ↴
      CHR(65, 66, 0x1F60B, 67, 68)

REM CODE converts the first character of
REM      a string to a code
PRINT "CODE "ABC" = "; CODE "ABC"
```

### 31.6.4 SPC (length)

Generates a string of spaces with the given length.

```
ASSERT (SPC(5) = "    ")
```

### 31.6.5 VAL(string)

The VAL function evaluates the string as a BC BASIC expression and return the value. For example, VAL("1 + 2") will return 3. The expression can use variables that you have set.

Note that VAL can be slow.

#### Example of the VAL function:

```
REM VAL will evaluate an expression  
a = 1  
b = 2  
PRINT "VAL ("a + b") = "; VAL ("a + b")
```

## 31.7 <STATEMENT> OVERVIEW

Statements are the building blocks of a BC BASIC program. They let you perform calculations, assign variables, loop until a condition is true, define functions and more.

#### Examples of statements:

```
10 CLS  
A = 3  
B = 4  
20 C = A + B  
PRINT "C is "; C
```

BC BASIC does not allow for multiple statements on a single line.

Statements can include an optional line number. They are the targets for GOTO and GOSUB statements. They do not need to be in any particular order. Other versions of BASIC always require line numbers and automatically order all lines by line number. BC BASIC instead lets you use any line numbers you want in any order.

Good line number practices:

1. Only number a statement when you have to.
2. Keep your line numbers in numerical order
3. Make your line numbers divisible by 10 or 100; that way you can add new line numbers in between existing values.

Line numbers in a function are *scoped* to that function; two functions can use each other's line number. You cannot GOTO or GOSUB into or out of a function.

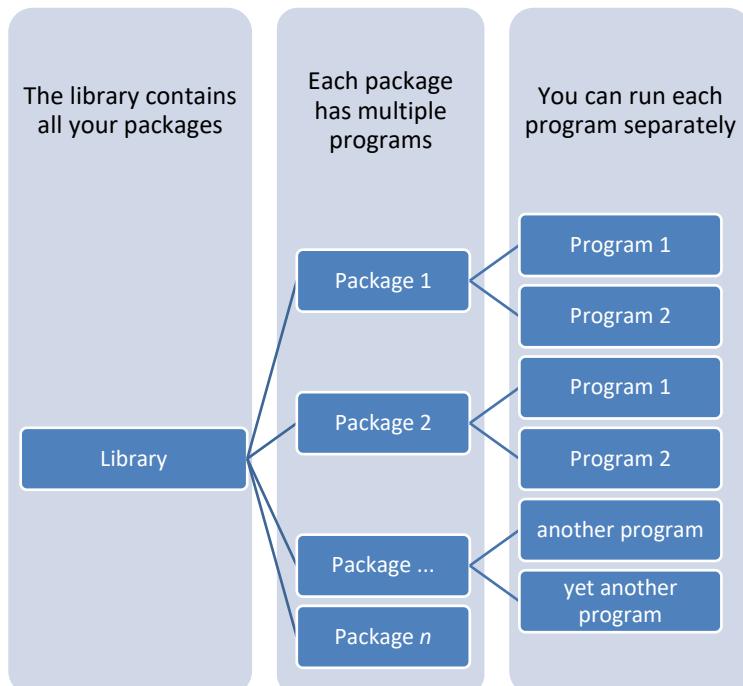
Statements are separated by new-lines; a newline is one or more of carriage-return, new-line, or vertical-tab. (Pressing ALT-newline in Word can separate lines with vertical tabs). You can continue a statement from one line to another by placing a visible enter symbol (↓, U+21B2) at the end of the line where white space would fit.

## 31.8 PACKAGES AND PROGRAMS

The BC BASIC *Package* and *Program* concepts are special to the BC BASIC environment.

If you're just getting started, you should feel free to just place the programs you write into the single package that you made when you wrote your first program. You don't even need to rename it; you can continue to use the New Package name.

But if you are going to write more than a few programs, you should spend a few minutes understanding the BC BASIC *Package* and *Program* concepts. These are explained more fully later on.



The BC BASIC environment tracks your programs for you. You do not have to deal with their file names or try to remember where the packages are or what programs they have. When you make a new package, the BC BASIC environment makes a file for you in the app roaming directory, and picks the name for you with a file extension of *.bcbasic*. Best Calculator will automatically read in all of the packages you've created (and they roam, too, so you can make

or edit a package on one computer and it will be automatically sent to your other computers).

### 31.8.1 Inside a package file

There is one file per package (and therefore multiple programs are placed into a single file). When Best Calculator starts, it reads in all the *.bcbasic* files both in the BC BASIC directory (these are the samples shipped with Best Calculator) and all the *.bcbasic* files in the app data directory (these are your packages).

## 31.9 PICKING A PACKAGE FOR YOUR PROGRAM

When you make a new program, you should add it to a package where it will logically fit. For your first few package, that will probably be the “New Package” that you created automatically when you wrote your first program. You will often find that you make a number of programs that you will be using together. For example, you’ve made one program that converts square feet to acres. The reverse program (acres to square feet) would naturally fit into the same package.

### 31.9.1 Creating common functions for several programs

The programs in a package are normally independent of each other. They don’t share variables or functions, and you can GOTO or GOSUB from one to the other. There are exceptions to this general rule. The IMPORT statement can read in the functions of another program in the same package. This lets you make a “library” of functions that many programs can use.

### 31.9.2 Exporting packages

You can export (write) packages out to a file and then later import (read) the files back in. When you export, you create a *.bcbasic* file that lists all the programs along with their names and descriptions and BC BASIC code. The export and import mechanisms mean that you can share code with your colleagues, coworkers, fellow students, or friends. You might even find useful packages on the internet. Be careful though: although a BC BASIC program cannot damage your computer, it’s possible for one package to delete your memory variables and conceptually possible to use up excessive disk space.

To export a package, pick its gear symbol and then select the  “save as” symbol. You then pick the folder and file to save to.

### 31.9.3 Pretty-print a package

Pretty-printing is like exporting but the resulting file is intended to be “pretty”. You can pretty-print a package as HTML, a Markdown, or as RTF (which is readable by Microsoft Word).

To pretty-print a package, pick its gear symbol and then select the  “pretty print” symbol. You then pick the output format: HTML, Markdown, or RTF (WORD).

Best Calculator BASIC produces marks code blocks in Githyb stylem surrounding the block of code with lines containing three back-ticks and the name of the language (BASIC) like this: ```BASIC.

Sample Markdown output in a markdown program:

## Markdown support in Best Calculator BASIC!

---

You can export your packages in markdown format! Markdown format is an increasingly popular format for creating computer documentation. From a markdown file you can documentation your project on popular blogs!

## Example Program

---

This is a BASIC "Hello World" program. Unlike old versions of BASIC, BC BASIC does not require every line to have a line number.

**Default Key:** ↵

```
REM Sample program
PRINT "Hello World"
```

### 31.10 ALL OF THE SPECIAL SYMBOLS

Best Calculator supports a number of special symbols both to help create good-looking program and so that your programs can be copy and pasted into word processors like Microsoft Word.

Symbol	Name	How it's used
$\sqrt{ }$	Square Root U+221A	$C = \sqrt{A^2 + B^2}$
$\sqrt[3]{ }$	Cube Root U+221B	$X = \sqrt[3]{Y}$
$\sqrt[4]{ }$	Fourth Root U+221C	$W = \sqrt[4]{X+Y}$
<sup>2</sup>	Superscript 2 U+B2	$C=X^2$
<sup>3</sup>	Superscript 3 U+B3	$C=X^3$
<sup>4</sup>	Superscript 4 U+2074	$C=X^4$
-	Hypen-Minus U+2D	The minus sign on a keyboard
–	Minus Sign U+2212	Alternate minus sign
–	En dash U+2013	Alternate minus sign
$\approx$	Approximately equal to U+2245	Helpful to compare floating point numbers.
$\not\approx$	Not approximately equal to U+2247	Also helpful for comparing floating point numbers
$\infty$	Infinity U+221E	Stands for an infinite amount
“ ”	Left and right double quotation marks U+201C and U+201D	Smart Quotes
$\backslash v$	Line Tabulation U+B	Can be used just like a normal carriage return (or Enter, Return or Line feed). Word sometimes converts those into a line tabulation
$\downarrow$	Downwards arrow with tip leftwards U+21B2	Line continuation: use at the end of a line to continue onto the next line.
$\downarrow$	Downwards arrow with corner leftwards U+21B5	Line continuation
$\downarrow$	Arrow pointing downwards then curving leftwards U+2936	Line continuation

## 32 BASIC STATEMENTS REFERENCE

In the descriptions of statements

- Words in <angle brackets> describe what to add.
- Words in square brackets are optional
- The punctuation ,... means the previous item can be part of a list separated by comma. The list can include no items at all.
- The punctuation ... means a series of statements on new lines
- Otherwise, the items are to be entered as they appear

For example, the CALL statement is described as **[CALL] <function> (<expression>, ...)**.

The word CALL is optional; you can include it or not as you please.

The <function> means you have to enter the name of a function

( ) are parenthesis; they are required

<expression> is an expression (they are explained in an earlier section)

The following are valid statements

```
CALL myfunction ()  
CALL myfunction (1)  
CALL myfunction (1+2)  
CALL myfunction (1, 2)  
myfunction ()
```

### 32.1 BEEP [DURATION, PITCH]

See also the PLAY command for fancier music capabilities.

Simples possible program that demonstrates BEEP:

```
BEEP
```

The BEEP command will BEEP the computer speaker briefly.

The DURATION option is the length to beep in seconds. The longest beep time is one second.

The PITCH option is a numeric value; 0 means middle C and 12 means a C notes one octave higher. Between are all the regular notes in a scale: C, C#, D, D#, E, F, F#, G, G#, A, A# and B. Numbers can be negative for lower octaves.

The duration and pitch must either both be present or both be absent; you can't specify one and not the other.

Example of BEEP with a duration and pitch

**BEEP .125, 0**

Makes a short beep that is middle C

**BEEP 1, -10**

A long, low-pitched beep

### 32.2 [CALL] <FUNCTION> (<EXPRESSION>, ...)

Calls the function by name, passing in the given parameters. The word CALL is optional, but the parentheses are required.

**Example of using CALL and defining a FUNCTION:**

CALL Hello("Mom")  
Hello("Dad")

FUNCTION Hello (name)  
PRINT "Hello ",name  
RETURN

The example will print Hello Mom and Hello Dad.

### 32.3 CLS [<COLOR>, <COLOR>] AND PAPER <COLOR>

CLS will clear the scrolling console and screen and potentially change the screen color. PAPER will change the screen color without clearing the screen. CLS also lets you set the foreground text color.

Normally after you press 'Run' the scrolling console will have the results of previous runs and will also have the results of evaluating your program. You can change the background color of the screen by specifying either a color name or a color index from the table below.

Pick the color for CLS and PAPER with either a color number (0 to 7) or a color name. In addition, color NONE can be used for transparent colors when making graphs using the Screen.Graphics() extension.

You can also set colors with HTML-like color strings. For example, **CLS #ADD8E6**, set the screen to the HTML "LightBlue" color.

Color Number	Color Name
0	BLACK
1	BLUE
2	RED
3	MAGENTA
4	GREEN
5	CYAN
6	YELLOW
7	WHITE

**Example of using CLS to clear the screen:**

```
CLS BLUE
```

**This example is like a color stroboscope:**

```
REM The screen only shows up if you PRINT
REM      something to it.
REM PAUSE delay is in "frames"; there are
REM      50 frames per second.
```

```
PRINT " "
delay = 25
FOR i=1 TO 3
FOR color = 0 TO 7
CLS color
PAUSE delay
NEXT color
```

```
REM You can specify colors with names
CLS BLACK
PAUSE delay
CLS BLUE
```

```
PAUSE delay
CLS RED
PAUSE delay
CLS MAGENTA
PAUSE delay
CLS GREEN
PAUSE delay
CLS CYAN
PAUSE delay
CLS YELLOW
PAUSE delay
CLS WHITE
PAUSE delay
NEXT i
CLS BLACK
```

Spring 2017 update: you can set the foreground color of CLS.

### 32.4 CONSOLE <EXPRESSION> [, <EXPRESSION>]

Writes the expressions out to the scrolling console. Any number of expressions can be given (including none)

**Example of writing to the console:**

```
CLS BLUE
ANGLE = 45
RADIANS = Math.DtoR (ANGLE)
```

```
REM PRINT to the screen to see the console.
PRINT ""
CONSOLE "SIN(45 degrees)", SIN(RADIANS)
```

The CONSOLE command prints out about 0.707. You might need to tap the

console key () to see the console. See '[Graphics and Best Calculator BASIC](#)' for a description of the console screen.

## 32.5 DATA – SEE THE READ AND DATA SECTION

Or, for the Data object that contains tables of information like cities and population, see the Data extension

## 32.6 DIM <NAME> ([SIZE [, SIZE]]) AND ARRAY METHODS

Creates an array variable. Use array variables when you need to store a set of values and then index them. You will often use arrays when you do data analysis. Arrays can be either 1-dimensional or 2-dimensional.

Arrays have several useful methods and properties like the Count property. These are described in the next section.

Example

```
DIM a()  
a(1) = "First"  
a(2) = "2.2"  
PRINT a(1)  
PRINT a(2)
```

You can find out the length of an array using the Count value.

```
FOR i = 1 to a.Count  
    PRINT a(i)  
NEXT i
```

You can also specify the size of the array in the DIM statement

```
REM Make an array that's exactly 10 items long  
DIM a(10)  
a(8) = 8.8  
  
REM Nope! You can't add an element beyond 10.  
a(20) = 20.20
```

The array also has a helper method array.AddRow(item1, item2, ...) that helps you make two-dimensional tables. These are often used when dealing with JSON data. You can call AddRow with any number of arguments. It will create a new array that contains all of the arguments and then will add the new array to the end of the array you called the method on.

Here's a real-world example showing how calculate the average of a list of number. The DIM statement isn't given a specific value, so the array can hold any number of elements.

```
CLS BLUE
DIM a()
REM Use DIM a(3) if you know the array will always
REM be 3 elements long. In that case, all three
REM elements will be initialized to NaN.
a(1) = 10
a(2) = 20
a(3) = 90

average = Average(a)
PRINT "Average is ";average
REM The average of 10, 20, 90 is 40

FUNCTION Average (data)
    sum = 0
    FOR i=1 TO data.Count
        sum = sum + data(i)
    NEXT i
    RETURN sum/data.Count
```

Example of adding a new value to the end of an array. In the example, an array called a is created. Two values are added to the end of the array.

```
DIM a()
a(a.Count+1) = "First new value"
a(a.Count+1) = "Second new value"
```

Example of creating a two-dimensional array that's an array of name/value pairs and then convert it to JSON

```
DIM list()
list.AddRow ("data", 82)
list.AddRow ("sensor", "Metawear")
json = String.Escape ("json", list)
print json
```

Examples of a two-dimensional array with a fixed size. It's initialized to zero.

```
DIM data(10,10)
data.Fill (0)
```

```
FOR i = 1 TO 10
FOR j = 10 TO 10
data[i,j] = i*10+j
NEXT j
NEXT i
PRINT data
```

The 2-D array is printed (it's truncated to help fit on the screen)

```
[[0,0,...,0,20],[0,0,...,0,30],...,[0,0,...,0,100],[0,0,...,0,110]]
```

## 32.7 DIM'D ARRAY METHODS AND PROPERTIES

When you DIM data(), you are making an array. These arrays have a number of useful properties and methods

### 32.7.1 Add(data1, data2, ...)

Adds the data as a column. For example

```
DIM data()
data.Add (1, 2, 3)
ASSERT (data[1] = 1)
ASSERT (data[2] = 2)
ASSERT (data[3] = 3)
```

The data array will have three rows, each with a values 1, 2, or 3.

### 32.7.2 AddRow(data1, data2, ...)

Adds the data as a row. For example

```
DIM data()
data.AddRow (1, 2, 3)
data.AddRow(4,5,6)
ASSERT (data[1,1] = 1)
ASSERT (data[1,2] = 2)
ASSERT (data[1,3] = 3)
ASSERT (data[2,1] = 4)
ASSERT (data[2,2] = 5)
ASSERT (data[2,3] = 6)
```

The data array will have three rows, each with a values 1, 2, or 3.

### 32.7.3 Clear() method

The Clear method removes all of the rows from the array.

### 32.7.4 Count property

The Count property tells you how many items are currently stored in an array

### 32.7.5 Fill (value)

Fills an array (either 1 or 2 dimensional) with a value. This is commonly used to zero out an array before using it. Arrays that are not zeroed out are filled with a NaN value.

### 32.7.6 HasKey(name) method

The HasKey (name) says if an array has a property with the given name. Arrays created by String.Parse("json", "{data string}") will have named properties that you can test with this method.

### 32.7.7 Max and Min properties

The Max and Min properties tell you the largest and smallest value in an array

### 32.7.8 MaxOf(column) and MinOf(column) methods

The MaxOf(column) and MinOf(column) methods tell you the largest and smallest value for a particular column of data in an array.

### 32.7.9 Mean method

Returns the mean (average) value of the array.

### 32.7.10 SetProperty method

Adds a new value to an array by name. If the named value isn't already in the array, the array will be expanded to include the new value; otherwise the old value is replaced by the new value.

This is used in the Christmas Music program to make a list of song names and their values. The AddSong method takes a title and the song notes and makes an array with a property Title and Notes. The small array is then added to the array of all songs.

In the example, list is an array created with **DIM list()**

```
FUNCTION AddSong (list, title, notes)
    DIM row()
    row SetProperty ("Title", title)
    row SetProperty ("Notes", notes)
```

```
list.Add (row)  
RETURN
```

When the program wants to print a list of all songs, it simply iterates through the list of songs, printing the title. After a song is selected, the song is played using the Notes property.

```
FOR i = 1 TO SongList.Count  
    song = SongList[i]  
    PRINT i, song.Title  
NEXT i
```

### 32.7.11 SumOfSquares method

Returns the sum of the square of each element. This is used, e.g., to calculate microphone (sound) volume

```
volume = SQR (data.SumOfSquares / data.Count)
```

### 32.7.12 Add() method and MaxCount and RemoveAlgorithm properties

These properties and the method help create fixed sized summaries of data.

You will add data to the array by calling the *array.Add(data)* method. If the data hasn't reached the MaxCount amount, the data is simply added to the end of the array. Otherwise, the array is full. The RemoveAlgorithm determines what happens next.

If RemoveAlgorithm is set to "First", the first array element is removed to make room and the latest data is added to the end.

If RemoveAlgorithm is set to "Random", a statistically chosen value (using [Reservoir sampling](#)) is possibly removed from the array to make room and, if a value was removed, the new data is added to the end. The data is removed in a way that every data point ever Add()'d to the array has the same chance of being in the array.

Here's a snippet from the "Hiking with an Altimeter" sample

```
DIM fullData()  
fullData.MaxCount = 200  
fullData.RemoveAlgorithm = "Random"
```

In this snippet, the fullData array is first created with the DIM statement. Then the fullData.MaxCount value is set to 200 so that only 200 data points are saved. Then the fullData.RemoveAlgorithm is set to “Random” so that the array will always be a reasonable summary of the overall data.

### 32.7.13 AddRow() method

The AddRow(data1, data2, data3, ...) method is a quick way to create a new array and add it to the end of an existing array. This will make an array-of-arrays.

The AddRow methods is commonly used when you’re creating JSON or CSV data files.

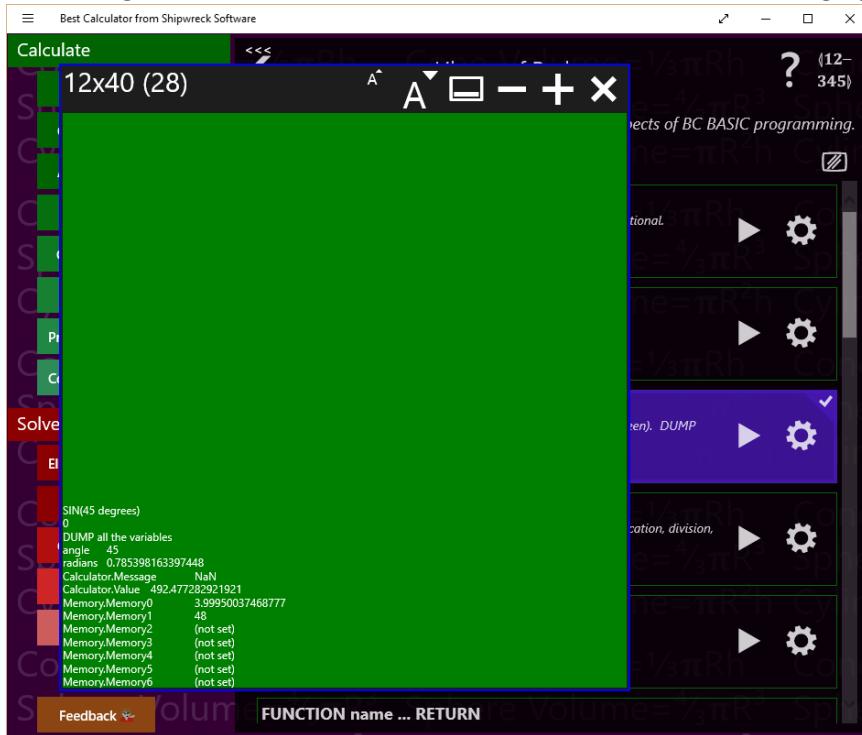
## 32.8 DUMP

Prints all the variables to the scrolling console. This is a common mechanism to see what your program is doing. DUMP will also print out all of the memory values.

### Example of using DUMP:

```
CLS BLUE  
ANGLE = 45  
RADIAN = Math.DtoR (ANGLE)  
  
REM PRINT to the screen to see the console.  
PRINT ""  
CONSOLE "SIN(45 degrees)", SIN(RADIAN)  
  
CONSOLE "DUMP all the variables"  
DUMP
```

The variables are then printed to the scrolling console.



You might need to press the console key (  ) to see the console.

### 32.9 FOR <VARIABLE> = <START> TO <END> [STEP <STEP>] ... NEXT <VARIABLE>

Use the FOR and NEXT statements to form loops. The variable is the name of the *loop variable*; it will start at the <start> value. Each time through the loop, it will be incremented by the <step> amount (the default is 1) until it's more than the <end> amount. The start, end and step values are all expressions.

If the <step> value is negative, then the loop is changed slightly. The variable starts at the start value, is decremented by the <step> value, and the loop ends when the variable is less than, not greater than the <end> value.

The end of the loop is the NEXT <variable> statement; the variable is the exact same as in the FOR statement. For example **FOR i=1 TO 10** is ended at a later **NEXT I** statement.

## Common errors:

1. Never GOTO or GOSUB into the middle of a FOR ... NEXT loop, and never jump out.
2. FOR ... NEXT loops can be nested inside each other, but always nest them correctly. The first FOR must match the last NEXT
3. You can reuse a variable name in several loops, but not nested.
4. You should not modify the variable inside the loop.
5. You will always go through the loop at least once

**Example of a simple FOR loop:**

```
PRINT "Value", "Squared"
FOR I=1 TO 10
PRINT I, I**2
NEXT I
```

The example prints a table of numbers and their squares.

Value	Squared
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

**Example of going backwards through a loop:**

```
PRINT "Value", "Squared"
FOR I=10 TO 1 STEP -1
PRINT I, I**2
NEXT I
```

To go backwards, you have to specify a negative STEP value and TO value that's less than the starting value. In the example, the STEP value is -1, and the TO value (1) is less than the start value (10).

**Example of a nested FOR loop:**

```
CLS GREEN
PRINT "Value **2 **3 **4 **5"
FOR N=1 TO 10
PRINT AT N+2,1 N
FOR POWER = 2 TO 5
PRINT AT N+2, (POWER-1)*8 N**POWER
NEXT POWER
NEXT N
```

This function prints a table of numbers; each number is printed along with the number raised to a power of 2, 3, 4 and 5.

Value	**2	**3	**4	**5
1	1	1	1	1
2	4	8	16	32
3	9	27	81	243
4	16	64	256	1024
5	25	125	625	3125
6	36	216	1296	7776
7	49	343	2401	16807
8	64	512	4096	32768
9	81	729	6561	59049
10	100	1000	10000	100000

## 32.10 FOREVER [WAIT|STOP]

The **FOREVER** command is an infinite loop; it will keep on processing forever. While it's processing, background events (like from IOT callbacks) will continue to be processed and the automatic objects (like the automatic graphs) will be updated several times per second. The FOREVER command can be stopped with the **FOREVER STOP** command.

This can be used in IOT scenarios where most of the processing happens in event callbacks. A button callback might be set up to call FOREVER STOP at which point the FOREVER statement will stop.

The FOREVER command is a simple convenience function. Some of the IOT examples use a FOR loop as a FOREVER statement and terminate the loop by setting the loop variable to a large amount.

FOREVER and **FOREVER WAIT** are equivalent statements. You can use either one and get the same effect.

See the IOT sample programs for examples of using FOREVER.

## 32.11 FUNCTION <NAME> ( <ARGS> ) ... RETURN [<VALUE>]

Creates a function with the given name (expressions are not allowed), taking the arguments. A single value can be returned.

When you CALL a function, you pass in data; that data is then given the names in the function. The names in the argument list are only valid in the function.

It's considered a best practice to have a single RETURN in a function; it's easier to understand a function that has only a single return point. However, it's also sometimes easier to RETURN early.

**Example of using FUNCTION to print to the screen:**

```
CALL Hello("Mom")
Hello("Dad")

FUNCTION Hello (name)
PRINT "Hello ",name
RETURN
```

In the example, the function HELLO will print whatever value is passed in. It's called twice, and therefore the program will print out two messages: Hello Mom and Hello Dad.

## 32.12 GLOBAL <VARIABLE>

The GLOBAL statement lets you to use global variables from within a function without having to pass the variable in as a parameter. For example, the callback functions used when a Bluetooth device has a changed characteristic value have no way to pass arbitrary data in and might need access to variables declared at the global level.

To use a global variable, use the GLOBAL <variable name> command inside your function.

Example

```
LET message = "This is a variable at the global scope"  
CALL PrintMessage ()  
STOP  
  
FUNCTION PrintMessage()  
    GLOBAL message  
    PRINT message  
RETURN
```

## 32.13 GOSUB <LINENUMBER> AND RETURN

*Tip: you should almost always use a function instead of using GOSUB. Many earlier versions of BASIC (including Sinclair BASIC and MSW BASIC) used GOSUB instead of FUNCTIONS.*

GOTO and GOSUB both jump to the line number indicated. GOSUB remembers where you jumped from. When the program encounters a RETURN statement, the program will continue at the line after the GOSUB line. You can nest a GOSUB inside a GOSUB routine.

It's considered good practice that each block of code that you will GOSUB to has a single entry point and usually will have just the one RETURN. Otherwise, your code will get very complex.

**Example of using GOSUB:**

```
REM Calculate the hypotenuse of a triangle
```

```
A = 3
```

```
B = 4
```

```
GOSUB 100
```

```
PRINT ""
```

```
DUMP
```

```
STOP
```

```
100 REM Calculate the hypotenuse from A and B
```

```
C=2 √(A**2 + B**2)
```

```
RETURN
```

### 32.14 GOTO <LINENUMBER>

Jumps to the line number indicated. Unlike GOSUB, you can't RETURN from a GOTO.

GOTO is often considered harmful.

In BC BASIC, the primary values of a GOTO is compatibility with earlier versions of BASIC and to help "extend" the statements in an IF statement. BC BASIC IF statements can only conditionally run a single statement after the THEN, and do not have ELSE clauses.

### 32.15 IF (<EXPRESSION>) THEN <STATEMENT> [ELSE <STATEMENT>]

If the expression is TRUE (not zero), then the statement will be run; otherwise it will not be. Often the statement will be a GOTO or GOSUB. The expression often uses the comparison operators (<>) plus AND OR and NOT).

**Example of an IF statement:**

```
a = 15
```

```
IF a > 12 THEN PRINT "A is more than a dozen"
```

In this example, the variable 'a' is set to the value 15. The PRINT part of the IF statement will only be executed if the variable a is greater than 12. Since 15 is

more than 12, the PRINT statement is executed, and “A is more than a dozen” is printed.

The statement after the ELSE is optional; it will be run if the expression is false.

```
IF (x >= 1) THEN PRINT "Single line: x >= 1" ↴  
ELSE PRINT "Else: x NOT >=1"
```

### 32.16 IF (<EXPRESSION>) ... [ELSE ...] ENDIF

The IF ... ENDIF statement is very similar to the IF statement but it lets you include multiple IF statements in a block instead of just one. It also lets you add statements in an ELSE clause. The statements in the ELSE clause will only be run if the expression is false.

```
IF (x < 1)  
    PRINT "Multi-line IF statement (expression is true)"  
    PRINT "x<1"  
ELSE  
    PRINT "Multi-line IF statement (expression is false)"  
    PRINT "NOT x<1"  
ENDIF
```

Not all statements can be placed inside of the statement blocks. Notably, you can't use GOTO statements.

### 32.17 IMPORT FUNCTIONS FROM “PROGRAM”

The IMPORT statement will read in all of the functions from a specified program in the same package. This lets you make a package with a common set of functions that all the programs in the package can use. This is useful when several programs in one package all need to perform the same calculation.

The STATISTICS samples do this. There's a program called “Sample Size Library” that consists of a set of useful statistical functions (MarginOfError, SampleSize, GetZ, and more). The programs that you might bind onto a key then just IMPORT the functions from that program and can call them.

Technical details: as needed, the library program is compiled and the functions remembered. When the IMPORT FUNCTIONS FROM “program” is run, the functions are imported by name. All functions are imported automatically. You can’t pick just one or two functions from a program.

You always IMPORT a program from the same package. You cannot IMPORT FUNCTIONS from a different package. You can IMPORT as many programs as you like; newer imported functions will override older ones. Deliberately doing this is not a best practice.

#### Example of the IMPORT statement:

```
IMPORT FUNCTIONS FROM "Conversion Library"
```

The example is taken directly from the AU to Meters program in the Astronomy library. As soon as it’s run, the program can call any function from the “Conversion Library” program.

### 32.18 INPUT <VARIABLE> [,<VARIABLE>...]

*Note: the expression <variable> = INPUT DEFAULT <value> PROMPT <prompt> is a more flexible and powerful way to read in data. The INPUT statement is included to improve compatibility with other versions of BASIC.*

The INPUT statement asks the user to enter a value. If the variable name ends with a \$ (dollar sign) the user may enter non-numeric values and the value will be a string value. Otherwise, the user can only enter a number.

You can request multiple values at once. For example: **INPUT amount, tax**

#### Example of the INPUT statement:

```
REM The a=INPUT expression has more power  
REM than INPUT statement. The expression version  
REM lets you specify a prompt and a default.
```

```
REM The INPUT statement has no default value  
REM and no prompt.  
INPUT taxrate
```

```
REM The INPUT expression has box a default  
REM and a prompt. The user has an easier time  
REM remembering what to enter.  
income = INPUT DEFAULT 40000 ↓  
PROMPT "Enter your income for the year"
```

```
PRINT "Owe="; taxrate*income  
IF (taxrate*income > 100) THEN ↴  
    PRINT "You owe more than 100"
```

### 32.19 (LET) <VARIABLE> = <EXPRESSION>

LET is the *assignment* statement. It sets (assigns) the value of the expression to a variable. The variable might or might not already exist. If the variable didn't already exist, it will be created. If it did exist, the old value is discarded and overwritten with the new value. Some languages call this an *assignment* statement.

Variables start with a letter, and then can be any combination of letters, digits, and underscores. Variables are case sensitive; myage is different from MYAGE and is different from myAge. Expression can include requests for user input.

The word LET is optional in BC BASIC. You will find that it is required in many other variants of BASIC.

#### Examples of the LET statement:

```
LET year = 2015  
birth_year = INPUT DEFAULT year - 15 ↴  
    PROMPT "When were you born?"  
age = year - birth_year  
PRINT ""  
DUMP
```

There are 3 LET statements in the example. The first uses the LET word (LET year = 2015). The other two leave off the LET word. When the INPUT expression is run, BC BASIC will pop up a dialog for the user to enter a value.

When were you born?

2000

OK

## 32.20 PAUSE <FRAMES>

Pauses the screen. This is useful when animating the screen. A value of 1 is one “frame”; there are about 60 frames per second. The value is not exact.

You can see an example of the PAUSE being used in the Colorful Countdown sample in the Quick Samples package.

## 32.21 PLAY <MUSIC>

The PLAY “music” statement will play the given music. The string must be in “Music Macro Language”; the simplest possible bit of music is simply **PLAY “C”** followed by **PLAY WAIT** which will play a middle C piano note and wait until the note has finished playing.

```
PLAY "C"  
PLAY WAIT
```

Wikipedia has a good article on MML at the link

[https://en.wikipedia.org/wiki/Music\\_Macro\\_Language](https://en.wikipedia.org/wiki/Music_Macro_Language). The PLAY command uses the modern version of the language (the difference is in things like how note lengths are specified).

Each note is one of CDEFGAB (or lowercase, cdefgab), possibly followed by a sharp sign (+ or # or \$) or flat (- or b). The octave is set with O or o <number> or > (one octave higher) or < (one octave lower). The L or l Change the note lengths with L or l <length> where length is 1, 2, 4, etc. for whole, half and quarter notes. Less common commands include p for pause, t <number> for tempo in beats per minute and v <volume> for volume,

You can pick different instruments to play the music. Best Calculator uses a MIDI file to sound the notes; the default instrument is an acoustic piano, but you can use any of the regular MIDI sounds. To pick a different instrument, use the “l” command. The **PLAY “I15 C”** command, for example, will play a middle C using Tubular Bells. A list of instruments is at

<https://www.midi.org/specifications/item/gm-level-1-sound-set>

If you create two PLAY commands, the second one will just be added to the end of the first. If the first is still playing, its default (like for the tempo and instrument) will carry over to the second. If the two are separated by a pause, the MIDI settings will be reset.

You can wait for the current music to end with **PLAY WAIT** and you can stop the current music with **PLAY STOP**.

### 32.21.1 Simple scales

```
REM Plays a scale over two octaves  
music = "CDEFGAB > CDEFGAB"  
PRINT "Play two scales " + music  
PLAY music  
PLAY WAIT
```

In this example, the music starts with middle “C” and continues up the scale. The “>” sign moves up an octave and the next octave up is played.

### 32.21.2 Switching Instrument with “I”

The “I” command is a BC BASIC extension to MML. It takes a numeric value which is a MIDI instrument number. “1” is an acoustic piano.

```
REM Play a scale with Tubular Bells  
music = "I15 CDEFGAB"  
PRINT "Play using Tubular Bells " + music  
PLAY music  
PLAY WAIT
```

### 32.21.3 PLAY ONNOTE “function”

Will call the given function every time a note is played. The function is called with parameters note (a MIDI note number), the instrument, the note duration in milliseconds and a string that represents the note in a user-friendly way. You can use this callback to animate your program.

### 32.21.4 PLAY STOP

The PLAY command will start to play music in the background. To stop the current music (for example, because you need to play something different), use the PLAY STOP command.

### 32.21.5 PLAY WAIT

The PLAY command will play music in the background; your program will continue to run even after the command and will run in parallel with the music. If you want to wait until the music is complete, use the PLAY WAIT command.

## 32.22 PRINT [AT ROW,COL] <EXPRESSION> [ (, OR ;) [AT ROW,COL] <EXPRESSION>]\*

The PRINT command will print one or more expressions to the output screen. BC BASIC remembers where the last thing was printed, and prints the next thing on the next line.

**PRINT expression** prints an expression (either a number or a string) onto the next line of the screen. This is the most common use of the PRINT command. If the screen is already full, nothing happens. The screen won't clear or scroll to make room for the new text.

**PRINT expression; expression** to print two values next to each other

**PRINT expression, expression** to print values in columns

**PRINT AT [row, col] for exact placement.** You can print each expression at a particular point on the screen with the AT row, col syntax. The rows and column values start with 1,1 at the upper-left corner of the screen. You can tell how large the screen is with the Screen.H and Screen.W values.

**Simplest example of PRINT:**

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

**Example of PRINT with an AT:**

```
PRINT AT 1,1 "HELLO"; AT 2,4 "WORLD"
```

**Example of doing a PRINT with multiple expressions and commas:**

```
PRINT "HELLO", "WORLD"
```

Note that there's a big gap between the "HELLO" and the "WORLD". The comma means to print at the next tab stop; those are each 16 characters apart.

**Example of doing a PRINT with multiple expressions and semicolons:**

```
PRINT "HELLO"; "WORLD"
```

With semicolons the two words are printed right next to each other without even a single space between them.

Spring 2017 update: the string to be printed can now contain any combination of embedded \r and \n characters. A \r\n is considered to be a single “newline” request, as are individual \r and \n characters. Strings from the Http.Get method often have these embedded characters.

Version 3.12: Orphan semicolons and commas will now influence the next PRINT statement. This enable the code from the classic Commodore one-liner to be ported to BC BASIC.

The one-line program from <http://10print.org> is:

```
10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

That program can be ported to BC BASIC with a few changes.

```
REM 10print.org 10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

```
CLS GREEN
FOR i = 1 TO 24*80
    REM 9585 = U+2571
    PRINT CHR$(9585.5+RND) ;
NEXT i
```

The changes are:

1. Always clear the screen
2. Use a FOR loop instead of looping forever and remove the line numbers.  
BC BASIC supports line numbers but usually you don't need them.
3. BC BASIC uses Unicode, not the Commodore character set. Unicode includes some nice slash characters in the Box-drawing character set.
4. RND isn't a function in BC BASIC (replace RND(1) with just plain RND)
5. BC BASIC doesn't support multiple statements per line

## 32.23 RAND <SEED> & THE RND VALUE

The RND value (like PI) acts like a variable in expressions; unlike PI (which is always the same value), RND will provide a stream of different random numbers between the values 0.0 and 1.0.

**Example of using RND:**

```
REM Print some random numbers  
PRINT RND, RND, RND, RND
```

This prints numbers like so:

(your numbers will be different!)

### Technical details

BC BASIC provides access to a single pseudo-random number generator; the stream of values is determined entirely by the value of the initial seed value. All expressions in all functions are connected to the same stream of random numbers. The RAND statement will re-seed the random number generator. The seed value 0.0 is special; it will re-seed the generator with a time-dependant value. All other seed values simply reset the seed.

*Tip: the random number generator is not suitable for any cryptographic or security uses.*

Comparison to other versions of BASIC:

BC BASIC allows you to use either RND by itself or RND([argument]) as a function. When no argument is present, or the argument is positive, it's just like

a RND value. If the value is zero or negative, it's like calling RAND to reset the random number generator. These variants are provided to be more compatible with other version of BASIC.

## 32.24 READ, RESTORE AND DATA

**READ m** will read data from the DATA statements into variable m. DATA 1,2,3,4 will create a set of data. Values in a data statement can be numbers or strings but not variables or expressions. READ and DATA are global; when a program starts all DATA statements will be found and added to the overall program state; as each READ happens (either in the global area or in a function) one item is taken from the global DATA statement. You can read multiple values at once:

**READ X, Y, Z** will read three values into variable X, Y and Z.

This little sample program will print 11 because the first data item is 11.

```
READ r
PRINT "READ", r
DATA 11, 22
DATA 333, 444
```

READ and DATA are often used in older programs.

The RESTORE statement will reset where the next READ will read data from. While READING data, you can use RESTORE; the next READ will then read from the first data item just as if no READ had ever been done. Unlike some versions of BASIC, in BC BASIC you can't specify which DATA statement to RESTORE to.

Once all data is consumed, the next READ will start again from the start. This is in contrast to some other version of BASIC where the next read will cause an error to occur.

Or, for the Data object that contains tables of information like cities and population, see the Data extension

## 32.25 REM COMMENT WORDS TO THE END OF THE LINE

The REM (remark) statement lets you put comments into your code. The words after the REM, and up to the carriage return, are entirely ignored by the program,

**Example of a REM comment statement:**

```
REM Calculate the hypotenuse given A and B  
C=2 √(A**2 + B**2)
```

The REM statement will help you understand what your program does. The common practice is to explain the *why* of a program, not the *how*. It's also common to comment on unusual or clever mathematical techniques. For example, some of the programs in the Real Estate section specifically refer to the exact regulations that are being implemented; this helps later on when you need to verify that the program is implemented in accordance with the laws.

## 32.26 SPEAK [VOICE <VOICE>] <TEXT>

BC BASIC includes a text-to-speech command that converts your input text into words.

The simplest program to demonstrate:

```
SPEAK "Hello World"
```

You should not give the SPEAK command long texts to read since there's no way to stop it! Speak will continue talking even after the BC BASIC program is complete.

The SPEAK command can use an optional *voice*. Different versions of Windows have different voices included. The SPEAK VOICES command will list all of the possible voices. On my main computer, the list of voices is

- Microsoft David Mobile
- Microsoft Zira Mobile
- Microsoft Mark Mobile

However, other computers will vary. You can pick a voice with just part of the voice name. For example, **SPEAK VOICE "David" "Hello, I am David"** will use the Microsoft David Mobile voice.

## 32.27 STOP [SILENT][VALUE] AND END

The STOP statement halts execution of the program. If rerun, the program will start from the very beginning again. It's common practice for programs to have their main logic at the start, and a number of functions at the end. A STOP statement is placed after the main logic, and before the subroutines.

BEST CALCULATOR will pop up a dialog box after a STOP statement and will display the given value. If no value is given, the last calculated value is displayed. The value will also be placed into the calculator display. The dialog is suppressed with STOP SILENT. Example: **STOP SILENT 1.1** will not display a value but will place 1.1 into the calculator display.

Older BASIC programs might use GOSUB-called subroutines instead of functions. GOSUB is inferior to functions.

### Example of using STOP to return a value to the calculator:

```
IMPORT FUNCTIONS FROM "Conversion Library"
```

```
from = Calculator.Value
m = ConvertToMeters(from, "au")
Calculator.Message = "Convert " + from ↓
    + " au into " + m + " meters"
STOP m
```

When the example is done, it returns the "m" value to the calculator. The calculator then enters the value into the display. The example is from the Space and Astronomy built-in package. It will not just work if you type it into a new program. To work, it requires that you have a program "Conversion Library" in your package.

END is permitted as a synonym for STOP. It is added for compatibility with other versions of BASIC including the first Dartmouth BASIC program (see the historical note).

## 33 EXTENSIONS REFERENCE

BC BASIC includes *extensions* to the core BASIC language. The extensions let you

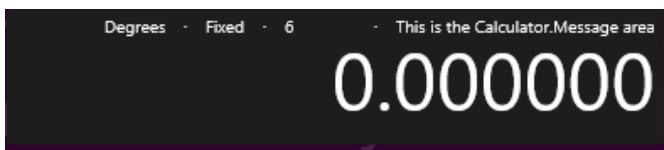
- Set and get values from the calculator screen
- Perform advanced math
- Set and get memory values that persist from one session to another
- Get information about the screen

### 33.1 CALCULATOR.VALUE AND CALCULATOR.MESSAGE EXTENSION

You can get and set the current numeric calculator value with the Calculator.Value value.

You can set and get the Calculator.Value; it is always a double. If you try to set it to a string value, the string will be converted to a double (e.g., the string “3.14” becomes the double 3.14; the string “apple” becomes the double NaN).

The Calculator.Message is the small display above the calculator value. When you start Best Calculator, it will say “System test passed”.



You can set (but not get) the Calculator.Message value.

**Examples of using the Calculator.Value and Calculator.Message extensions:**

```
value = Calculator.Value
retval = value * Math.PI
Calculator.Message = "Converted diameter " +
    + value + " to circumference"
Calculator.Value = retval
```

First the variable called “value” is set to the current value in the calculator window. It’s multiplied by PI. Then a message is printed to the calculator window, and the calculator value is set to the variable called “retval”.

In addition you can get the exact set of digits on the screen with `Calculator.ValueString`. This can be used with, e.g., the `SPEAK` command.

## 33.2 DATA EXTENSION

The Data extension contains tables of useful information.

### 33.2.1 Data location values

The Data object includes a summary of place names from around the world. Cities must have a population of at least 100,000 people to be included in the list.

The city objects returned include several name field (**Name**, **FullName**, **GeoNameId**, **Admin1** and **Country**), location (**LatitudeDD**, **LongitudeDD** and **Elevation**, where the DD stands for “degrees decimal”) plus the city **Population**. Data is from a 2018 almanac.

### 33.2.2 Data.GetLocations (“name”)

The `Data.GetLocations (“name”)` will return a list of all cities whose name matches the input name. If you pass in “york”, “New York City” and “York” (UK) will both be returned along with several other places whose name includes York around the world.

Example:

```
CLS BLUE WHITE
PRINT "Demonstrate Data.GetLocations"
PRINT "Selects multiple locations and show useful
information"
cities = Data.GetLocations("york")
IF (cities.Count = 0)
    PRINT "No locations were found"
    STOP
END IF

PRINT ""
FOR i = 1 TO cities.Count
    city = cities[i]
    PRINT city.FullName; " has a population of ";
    city.Population
NEXT i
```

The output of the example program lists 5 matching cities

**Demonstrate Data.GetLocations**

Selects multiple locations and show useful information

East New York (NY, US) has a population of 173198

East York (08, CA) has a population of 115365

New York City (NY, US) has a population of 8175133

North York (08, CA) has a population of 636000

York (ENG, GB) has a population of 153717

**33.2.3 Data.Picklocation()**

The Data.PickLocation() method will pop up a small dialog and let the user select a location. The return will be a city object with the same set of fields as the Data.GetLocations("name") method.

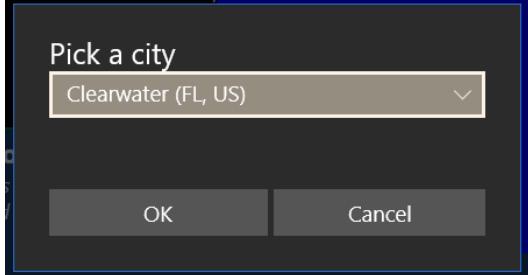
**Example:**

```
CLS BLUE WHITE
PRINT "Demonstrate Data.PickLocation"
PRINT "Select a location and show useful information"
city = Data.PickLocation()
IF (city.IsError)
    PRINT "No location was selected"
    STOP
END IF
```

The example then continues and prints the city details

```
PRINT "Name", city.Name
PRINT "Elevation", city.Elevation
PRINT "Population", city.Population
PRINT "Latitude", city.LatitudeDD
PRINT "Longitude", city.LongitudeDD
PRINT ""
PRINT "Name Details"
PRINT "GeoNameId", city.GeoNameId
PRINT "FullName", city.FullName
PRINT "Admin1", city.Admin1
PRINT "Country", city.Country
```

**The dialog looks like this:**



The output for Clearwater is

```
Demonstrate Data.PickLocation
Select a location and show useful information

Name           Clearwater
Elevation      15
Population    113003
Latitude       27.96585
Longitude     -82.8001

Name Details
GeoNameId      4151316
FullName       Clearwater (FL, US)
Admin1          FL
Country         US
```

### 33.3 DATETIME EXTENSION

The DateTime extension is designed to let you add time stamps to your date.

#### 33.3.1 DateTime.UtcNow() and DateTime.GetNow()

The DateTime.UtcNow() method returns a filled-in DateTime object set to the current time. GetNowUtc is similar but returns a DateTime with the current UTC (universal time, almost identical to GMT, Greenwich Mean Time).

#### 33.3.2 DateTime.Parse ("4/12/2019")

Creates a new DateTime object by parsing a date-formatted input string. Many common date formats are supported.

Example code:

```
REM Demonstrate how to use the DateTime.Parse
function
```

```
CLS
```

```
PRINT "Input", "Date", "Time"
```

```
Demo ("4/17/2019")
```

```
Demo ("4-17-2019")
```

```
Demo ("17-APR-2019")
```

```
Demo ("17/April/2019")
```

```
Demo ("4/17")
```

```
Demo ("2019-04-17T00:00:00.000000-07:00")
```

```
Demo ("Wed, 17 Apr 2019 07:00:00 GMT")
```

```
Demo ("Wed, 17 Apr 2019 07:00:00")
```

REM Trivial little function takes in a string, converts it  
 REM to a DateTime with DateTime.Parse, and prints  
 REM the original string and the resulting date and time.

```
FUNCTION Demo (str)
```

```
    dt = DateTime.Parse(str)
```

```
    PRINT str, dt.Date, dt.Time
```

```
RETURN
```

This will print out the following results

Input	Date	Time
4/17/2019	2019-04-17	00:00:00.00
4-17-2019	2019-04-17	00:00:00.00
17-APR-2019	2019-04-17	00:00:00.00
17/April/2019	2019-04-17	00:00:00.00
4/17	2019-04-17	00:00:00.00
2019-04-17T00:00:00.000000-07:00	2019-04-17	00:00:00.00
Wed, 17 Apr 2019 07:00:00 GMT	2019-04-17	07:00:00.00
Wed, 17 Apr 2019 07:00:00	2019-04-17	07:00:00.00

### 33.3.3 DateTime.Add (years, months, days, hours, minutes, seconds)

Given an existing DateTime object, adds the number of years, months, etc. The values can be more than you might expect: for example, you can add 86400 seconds instead of adding a day.

```
REM Whats the time in 5 minutes?  
dt = DateTime.GetNow()  
PRINT "Starting time", dt.Time  
  
REM Add 5 minutes.  
REM The parameters are year, months, days,  
REM hours, minutes and seconds.  
dt.Add (0, 0, 0, 0, 5, 0)  
PRINT "Final Time", dt.Time
```

### 33.3.4 DateTime.Set (year, month, day, hour, minute, seconds)

Given an existing date, sets the date and time to the values given.

### 33.3.5 DateTime.Subtract (datetime)

Use the DateTime.Subtract(datetime) method to get the number of seconds between two DateTime values. This is used when you need to know how much time has elapsed.

Example:

```
startTime = DateTime.GetNow()  
REM now perform some calculations  
endTime = DateTime.GetNow()  
delta = endTime.Subtract (startTime)  
PRINT "That took ", delta, " seconds"
```

The EX: DateTime example programs includes a full program to demonstrate how to use DateTime to measure your program performance.

### 33.3.6 Hour, HourDecimal, Minute, Second, Year, Month, MonthName, Day, DayOfWeek, and DayOfYear properties

DateTime includes a number of properties to get the current date and time split into the component parts (hour, minute, etc.).

The MonthName will be in the user's current preference.

HourDecimal is the current hour, minute and second as a decimal value. For example, 1 hour 15 minutes is 1.25 HoursDecimal.

### 33.3.7 DateTime.Date, DateTime.Time and DateTime.TimeHHmm

The Date and Time properties will return the date and time set when you called GetNow(). The values will be returned as strings which are designed to be put into a CSV file as a timestamp and interoperate with Excel. The DateTime.TimeHHmm is designed for Excel files where you want to show the hours and minutes more clearly (when you make a CSV file where the time stamps include seconds and milliseconds, Excel will not display the hours by default),

An example DateTime.Date is “2017-03-23” (March 23<sup>rd</sup>, 2017)

An example DateTime.Time is “13:55:23.34” (1:55 PM and 23.34 seconds)

An example DateTime.TimeHHmm is “13:55” (1:55 PM)

The Date and Time are similar to RFC 3339 dates and times.

### 33.3.8 DateTime.Iso8601 and DateTime.Rfc1123

The DateTime object knows about two common date/time formats used on the internet. Iso8601 is an internet standard; it’s also the date standard preferred by XKCD in <https://xkcd.com/1179/>. In addition, it’s the format preferred for JSON data.

Format	Example
Iso8601	2017-04-16T07:31:56.9603069-07:00
Rfc1123	Sun, 16 Apr 2017 14:31:56 GMT

Although the Rfc1123 format is required by many internet standards, it has some issues: it’s a very complex format that’s designed to be extra human readable, but only for humans that read English. Using this format in new standards is not recommended.

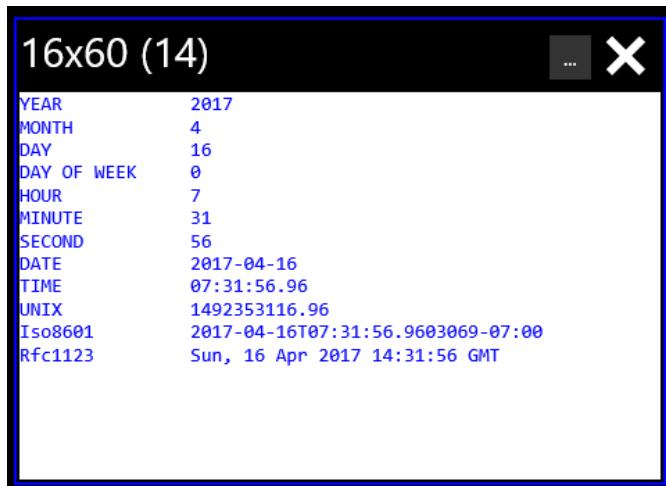
### 33.3.9 DateTime.AsTotalSeconds

The AsTotalSeconds returns a number (not a string) of the number of seconds the DateTime represents. The value uses Unix Time as the basis (the number of seconds since 1970-01-01, January 1<sup>st</sup>, 1970).

### 33.3.10 Example of common properties

You can use all of the properties to make a simple clock display. This example demonstrates getting the current time and displaying all of the component parts.

```
now = DateTime.GetNow()  
  
PRINT AT 1,1 "YEAR", now.Year  
PRINT AT 2,1 "MONTH", now.Month  
PRINT AT 3,1 "DAY", now.Day  
PRINT AT 4,1 "DAY OF WEEK", now.DayOfWeek  
PRINT AT 5,1 "HOUR", now.Hour  
PRINT AT 6,1 "MINUTE", now.Minute  
PRINT AT 7,1 "SECOND", now.Second  
PRINT AT 8,1 "DATE", now.Date  
PRINT AT 9,1 "TIME", now.Time  
PRINT AT 10,1 "UNIX", now.AsTotalSeconds
```



In the example, the code was run on 2017-04-16 at 7:31 in the morning.

## 33.4 FILE EXTENSION

The File extension lets you read, write and append files. It includes pickers so that you can pick a file to read, write or append.

There are three types of pickers: the ReadPicker() selects a file that you can read from, AppendPicker() picks a file that will be appended to, and WritePicker() picks a file that will always be overwritten.

### 33.4.1 file=File.AppendPicker(), file.AppendLine, file.AppendText() and file.Size

Starting with the File object you can ask the user to pick a file to append to and then append data to the file. This is similar to the WritePicker and WriteText methods except that the Append methods will add to the existing file and the Write methods will erase the file first.

Note: there's a problem with the Windows runtime: the picker will properly open the file for appending. However, you'll get a popup asking if it's OK to overwrite the file. The file will not be overwritten.

Example of using AppendPicker to write CSV data to a file. If the file starts off empty, headers are added

```
REM
REM Demonstrate AppendPicker, AppendText and
AppendLine
REM

file = File.AppendPicker("CSV file", ".csv", "test.csv")
IF (file.IsError)
    REM file will have a error message
    PRINT file
    STOP
END IF
PRINT "SIZE", file.Size()
IF (file.Size( ) = 0) THEN file.AppendLine("time,data")
now = DateTime.GetNow()

REM
REM Use an array to make
REM perfect CSV data
REM
DIM data(2)
data(1) = now.Time
data(2) = 42.42
file.AppendText (String.Escape("csv", data))
```

### 33.4.2 file=File.ReadPicker(".txt") , file.ReadAll(), file.ReadLines()

Starting with the File object you can ask the user to pick a file to read, and then either read the file as one large text or read the file as lines.

File is only available to Best Calculator, IOT Edition.

File.ReadPicker() will return a file that the user selects. Once you have this file you can ReadAll() to read the entire file and ReadLines() to read the file as a set of lines. You can also get the Size() of the selected file.

If the user did not pick a file, file.IsError will be true.

Once a file is read, you can parse CSV (Comma-separated values) and JSON (JavaScript Object Notation) into arrays. Use the String.Parse() method to convert the text into an array of data.

Examples:

```
REM
REM Demonstrate the File.ReadPicker
REM

CLS GREEN
PRINT "Demonstrate reading a file"
file = File.ReadPicker ("*.txt")
IF (file.IsError)
    REM file has an error message
    PRINT file
    STOP
END IF
PRINT "Size is ", file.Size()

REM ReadAll will read the entire file as single text.
fulltext = file.ReadAll()
PRINT "The entire file"
PRINT fulltext
PRINT ""

REM
REM ReadLines will read the entire file and split it
REM into individual lines.
REM
lines = file.ReadLines()

PRINT "Count of lines", lines.Count
IF (lines.Count > 1) THEN PRINT "First line", lines[1]
```

You can also provide a list of extensions that you are willing to accept.

```
DIM extensions()
extensions(1) = ".json"
extensions(2) = ".txt"
extensions(3) = ".csv"
CLS BLUE
PRINT extensions.Count
file = File.ReadPicker(extensions)
PRINT file
```

### 33.4.3 file=File.WritePicker(), file.WriteLine, file.WriteAllText() and file.Size

Starting with the File object you can ask the user to pick a file to write to. This is similar to the AppendPicker and AppendText methods except that the Write methods will always overwrite the entire file.

Example of using WritePicker to write CSV data to a file. Unlike in the Append example, the file is always completely overwritten.

```
REM
REM Demonstrate WritePicker, WriteText and WriteLine
REM

file = File.WritePicker("CSV file", ".csv", "test.csv")
IF (file.IsError)
    REM file will have a error message
    PRINT file
    STOP
END IF

file.WriteLine("time,data")
now = DateTime.GetNow()

REM
REM Use an array to make
REM perfect CSV data
REM
DIM data(2)
data(1) = now.Time
data(2) = 42.42
file.WriteAllText(String.Escape("csv", data))
```

### 33.5 GOPHER EXTENSION FOR GOPHER-OF-THINGS

The Gopher internet protocol is lets you publish an internet endpoint with menus and data. Users just need a Gopher program to view the menus. Your BC BASIC program will completely control what's displayed.

To use the Gopher server library, you need to set up three things

1. You need to create callback functions that return the Gopher menus that you design. The gopher menus are what the user will see in their gopher client.
2. You need to create *routes* to map incoming Gopher request to the individual functions that you made. This is done using gopher *selectors* which are just strings. A gopher client always starts off asking a gopher server for a blank selector, so you should always set up a route for at least the blank selector.
3. You need to start the Gopher server, giving it a name that will be broadcast via DNS-SD (only a few Gopher clients, including the Windows Simple Gopher Client, understand DNS-SD).

#### 33.5.1 Gopher callback functions and Gopher menus

A Gopher callback function is a function you write that takes in three parameters. The parameters are a selector, a list of values, and the search string. The Gopher server will call it when an incoming Gopher internet request is seen.

A Gopher callback function must return a Gopher menu item; the menu will be sent back to the original internet request.

Here's an example of a very simple Gopher callback function that returns a menu with a series of "i" (information) elements with the current time in different formats.

```
FUNCTION GOPHER_TIME(selector, ids, search)
now = DateTime.GetNow()
REM Create the menu to return
REM Type "i" is information
REM Type "1" is a new directory (menu)
REM The Time menu is sometimes called as inline,
REM so make sure that the first line returned is
REM useful information.
```

```
menu = Gopher.NewMenu()
menu.Add ("i", "ISO: " + now.Iso8601)
menu.Add ("i", "RFC 1123: " + now.Rfc1123)
menu.Add ("i", "Seconds: " + now.AsTotalSeconds)
menu.Add ("i", "")
menu.Add ("1", "GOTO MAIN", "")
RETURN menu
```

In this Gopher callback function (from the Gopher example code), the menu is created by calling Gopher.NewMenu(). Menu entries are added one by one.

### Gopher Menu Add method parameters

The parameters to the Gopher Menu Add() method are

- **Menu Entry Code**, a one-character code that tells the final Gopher client how to handle the menu entry. See the table below for common Gopher menu entry codes
- **User string** which is what's displayed to the user
- **Selector** which will be sent back to the Gopher server when the link is selected.
- **Host**, the name of the computer to send any commands to. If you leave this blank, it will be filled in automatically with the IP address of the computer that send the original Gopher command
- **Port**, the port number to send any commands to. Like the host, if you leave this blank ("") it will be filled in automatically
- **Options**, not part of the Gopher standard. If you pass in INLINE and the Gopher client is the Simple Gopher Client app, then any directory calls will be “inlined” instead of creating a new menu

#### 33.5.2 Common menu entry types

There are over twenty menu entry types that Gopher clients need to be able to handle. Of those, your Gopher programs for the BC BASIC Gopher Server will commonly only these three.

Menu Entry Code	What it means
1	Directory. You will also need to provide a selector. You will often not need to provide a host or port.

h (lower-case)	HTML. The selector will be a URL that the Gopher client will jump to. You will not need to provide a host or port.
i (lower-case)	Info. The results will be printed as text. You will never need to provide a host or port because the info type is not a clickable or selectable item.

The Gopher protocol includes menu entry types for file downloads and images, but the BC BASIC Gopher programming model does not allow you to reply to those kinds of requests.

### 33.5.3 Selector versus ids

Your Gopher callback function should carefully pick between using the selector to make decisions versus using the `ids` object. This choice is based in part on the route you set up (and which is explained further down).

Let's say you want a particular Gopher callback function to set the color of a light. You might set up a selector like this (from the "Gopher of Things" example program):

```
Gopher.AddRoute ("/setRGB/{id}/{red}/{green}/{blue}",
"GOPHER_LIGHT")
```

This route means that as long as the Gopher client program asks for a page where the selector starts with /setRGB/ and includes four values, the function `GOPHER_LIGHT` will be called. An actual selector from the Gopher client might look like `/setRGB/1/255/0/0` .

When the `GOPHER_LIGHT` function is called, the passed-in selector will be the original selector. The `ids` object will have four values: `id`, `red`, `green`, and `blue`. You can use those values directly as demonstrated in this little code snippet from the same example:

```
GLOBAL devices
dotti = devices[ids.id]
dotti.SetPanel (ids.red, ids.green, ids.blue)
```

The code is directly using the values from the selector without you having to carefully parse them out; BC BASIC has already done this for you!

### 33.5.4 Gopher.AddRoute (“route”, “function”)

The Gopher.AddRoute () method lets you connect an incoming Gopher request’s selector with a BC BASIC function that you create that will handle that request. The first parameter is the selector to match;

The selector will be matched in a case-sensitive way. A route for “/SET” won’t match an incoming selector “/set”. The routes length (when split by the ‘/’ character) must also always match the incoming selector from the Gopher client. For example, the route /person/me won’t match an incoming actual selector “/person” or “/person/me/verbose”. It will only match “/person/me”

The route can include a sort of variable syntax. A route segment (between the slashes) can be surrounded by curly braces (“/person/{name}”). The {name} will match any incoming selector. For example, it will match “/person/erik”. The name erik will be added to the ids list that’s passed into the Gopher callback function as “name”.

### 33.5.5 Gopher.Start (“name”)

The Gopher.Start(“name”) method will start the gopher server on port 70.

The name is the broadcast DNS-SD name; the full DNS-SD name that will be used is name.\_gopher.\_tcp.local. Some Gopher client programs (like the Simple Gopher Client on the Windows app store) are able to find Gopher servers based on the DNS-SD broadcasts.

The port is currently set at 70 and cannot be changed. Once the Gopher server has started, it cannot be stopped.

### 33.6 HTTP EXTENSION

The Http extension lets you read and write data from and to web services.

The Http extension is only available in Best Calculator, IOT edition.

A complete example of writing to a web service is in the Complete Example section, Connecting to Microsoft Flow .

#### 33.6.1 Http.Get(url, [headers])

Download data from the internet using Http.Get(). You pass in a URL (Uniform Resource Locator) and an optional array of headers. The result will either be an error or a filled-in structure with the content plus the HTTP status code and reason (which will often just be the string OK)

In the example, data is downloaded from a news feed and then

1. It's checked to make sure the download was OK
2. Some data about the download is printed
3. The Http content is parsed as JSON (JavaScript Object Notation) and data is pulled out

Example: REM Demonstrate downloading from the internet

```
REM
REM Download content from a news feed
REM Make sure the download was OK
REM Parse the JSON into data
REM

url = "https://hacker-
news.firebaseio.com/v0/item/8863.json?print=pretty"
result = Http.Get (url)
IF (result.IsError)
    REM Did not get data
    CLS RED
    PRINT "Unable to download URL"
    PRINT "ErrorCode", result.ErrorCode
    PRINT "ErrorMessage", result.ErrorMessage
ELSE
    REM All OK
    CLS GREEN
    PRINT "Downloaded from URL"
    PRINT "Status", result.StatusCode
    PRINT "Reason", result.ReasonPhrase
    PRINT "Content", result.Content
```

```
REM Now parse it as json  
REM You can pull individual bits out  
data = String.Parse("json", result.Content)  
PRINT "data.by", data.by  
PRINT "data.title", data.title  
END IF
```

### 33.6.2 Http.Post() and Http.Put()

The `Http.Post` (`url`, `content`, `[header]`) and `Http.Put` (`url`, `content`, `[header]`) work the same way except for the HTTP method. The `Post` method will send data with the HTTP POST verb and the `Put` method will send data with the HTTP PUT verb.

The content for both is a string, and the header is an optional array of strings that will be parsed into HTTP headers. Note that the headers must be in the correct formats; one of the most common errors is to misspell HTTP headers.

The example is a single function taken from the larger Microsoft Flow example. In the example, data is put into an array and then converted into a JSON-formatted string. A Content-Type header is also created and then the `url`, `content` and the `header` are used in an `Http.Put(url, content, header)` call.

```
REM  
REM Format and send data to Microsoft Flow  
REM  
FUNCTION SendData(url, data, time, deviceName, sensor,  
min, max)  
REM  
REM Put the data into correct JSON form  
REM  
DIM datalist()  
datalist.AddRow ("data", data)  
datalist.AddRow ("time", time)  
datalist.AddRow ("device", deviceName)  
datalist.AddRow ("sensor", sensor)  
datalist.AddRow ("min", min)  
datalist.AddRow ("max", max)  
json = String.Escape ("json", datalist)  
  
PRINT json  
  
REM Microsoft Flow demands that data be passed  
using the  
REM a Content-Type of application/json.  
DIM header()  
header[1] = "Content-Type: application/json"
```

```
result = Http.Post (url, json, header)
```

```
RETURN result
```

## 33.7 MATH EXTENSION

BC BASIC includes a full set of math functions and constants divided into categories for trigonometry, rounding, logarithm and power functions and other functions.

### 33.7.1 Trigonometry (Math.Sin (radians) and more)

BC BASIC does all trigonometry calculations in radians. The function Math.DtoR (degrees) will convert degrees to radians and Math.RtoD(radians) will convert radians to degrees.

Function	Notes
Math.Acos(value)	Calculates the inverse of Math.Cos; given a value will compute the corresponding angle in radians.
Math.Asin(value)	Calculates the inverse of Math.Sin; given a value will compute the corresponding angle in radians.
Math.Atan(value)	Calculates the inverse of Math.Tan; given a value will compute the corresponding angle in radians. Note that Math.Tan of 90° is infinite.
Math.Atan2(y, x)	Calculates the inverse tangent of given an Y and X value. Note that Y is given first. This matches most common program languages including C#, Java, Fortran, C and JavaScript. Unlike the Math.Atan function, Math.Atan2 can handle angles of 90°
Math.Cos (radians)	Calculates the cosine of an angle given in radians
Math.Cosh (radian)	Calculates the hyperbolic cosine of an angle given in radians
Math.DtoR (degrees)	Converts degrees to radians
Math.RtoD (radians)	Converts radians to degrees
Math.Sin (radians)	Calculates the sin of an angle given in radians
Math.Sinh (radians)	Calculates the hyperbolic sin of an angle given in radians
Math.Tan (radians)	Calculates the tangent of an angle given radians
Math.Tanh (radians)	Calculates the hyperbolic tangent of an angle given radians

### 33.7.2 Rounding and sign (Floor(), Round() and more)

Function	Notes
Math.Abs (value)	Calculate the absolute value of a number.
Math.Ceiling (value)	Calculates the ceiling of a number. The ceiling is the number rounded up to the nearest integer. For example, Math.Ceiling (2.2) is 3. Ceilings of negative numbers round up (e.g., to be closer to zero), so Math.Ceiling (-2.2) is -2.
Math.Floor (value)	Calculates the floor of a number. The floor is the number rounded down to the nearest integer. For example, Math.Floor (2.8) is 2; Math.Floor (-2.8) is -3.
Math.Frac (value)	Returns the fractional part of a number (the part after the decimal sign). Math.Frac(3.456) is 0.456.  For negative numbers, Math.Frac returns the difference between the number at the next higher number. For example, Math.Frac(-7.4) is 0.6.  Math.Floor(value) + Math.Frac(value) is equal to the original value.
Math.Max (value, ...)	Returns the largest value of a set of numbers. You may give one or more values to Math.Max()
Math.Min (value, ...)	Returns the smallest value of a set of numbers. You may give one or more values to Math.Max()
Math.Mod (v1, v2)	Returns the remainder when v1 is divided by v2. For example, Math.Mod(7,3) is 1 because 3 goes into 7 2 times with a remainder of 1. Math.Mod(7.6, 3.1) is 1.4 because it's the remainder after 3.1 is multiplied by 2.
Math.Round (value [,dp])	Calculates the rounded value of a number. The rounded value is the one closest to an

	<p>integer. Math.Round (2.2) is 2; Math.Round (2.8) is 3. If a number is a “.5” number, it is rounded down (technically, rounded towards zero; Math.Round (2.5) is 2, and Math.Round (-2.5) is -2.)</p> <p>If the dp (decimal places) value is given, it's the number of decimal places to round to. For example, Math.Round (1.234, 2) will return 1.23. The default is zero, meaning round to an integer. You can pass in negative values. For example, Math.Round (1234, -2) will return 1200.</p>
Math.Sign (value)	Return the sign of a number. The sign is 1 for positive values, -1 for negative values, and 0 for zero.
Math.Truncate (value)	Calculates the truncated value of a number. The truncated value is the integer value closest to zero. For positive numbers, this is like Math.Floor (for example, Math.Truncate (2.8) is 2). For negative numbers, this is like Ceiling (for example, Math.Truncate (-2.8) is -2, the integer closer to zero)

### 33.7.3 Bit functions Math.BitAnd, Math.BitOr, and Math.BitNot

Function	Notes
Math.BitAnd(value1, value2)	Calculates the bitwise OR of two values as if they were 32-bit unsigned integers.
Math.BitOr (value1, value2)	Calculates the bitwise OR of two values as if they were 32-bit unsigned integers.
Math.BitNot (value)	Calculates the bitwise inverse of the given value as if it was a 32-bit unsigned integer.

### 33.7.4 Logarithm and power functions (Math.Log, Math.Exp, and more)

Function	Notes
Math.Exp(value)	Calculates the value $e^{\text{value}}$ for any given value. This is the reverse of the Math.Log function
Math.Log (value) Math.Log (value, base)	Can do two different calculations. When given just one value, calculates the natural (base e) logarithm of the given value. When given two numbers, calculates the $\log_{\text{base}}$ of the value.
Math.Log2 (value)	Calculates the base-2 logarithm of a number. This is useful when dealing with computer math. Simple example: you're writing a program, and certain variable will hold a number from 0 to 934. How many bits are needed to hold this value? Answer: Math.Log2(934) is about 9.87; rounding up, you discover than you will need a 10-bit field to hold the number.  Sophisticated example: You need to store 200 numbers, each of which is a value 0 to 11 (inclusive, 12 total values). Assuming best bit packing but no compression, how much space do you need? The answer is that $200 * \text{Math.Log2}(12) = 717$ bits; divide by 8 to discover that your data will fit into 90 bytes of space.
Math.Log10(value)	Calculates the base-10 logarithm of a number. This is useful when rounding a number up to the nearest power-of-ten.

	For example, you want the first power of ten (e.g., 10, 100, 1000) of 783. Math.Log10 (783) is 2.89; rounded up this is 3. Math.Pow (10, 3) is 1000, and that's the closest larger power of ten of the number.
Math.Pow (x, y)	Calculates the exponent $x^y$ . For example, Math.Pow (10, 3) is 1000.
Math.Sqrt(value)	Calculates the square root of the value.

### 33.7.5 Math.Factorial and Math.IsNaN

#### The Math.Factorial Function

Math.Factorial(value) is the  $n!$  function. For any given integer, it returns the product of all the integers less than or equal to  $n$ . For example, Math.Factorial(5) is  $5 * 4 * 3 * 2 * 1$ , or 120.

Math.Factorial returns NaN (not a number) for any input that isn't an integer or is less than zero. Math.Factorial(0) is 1.

#### Example of using Math.Factorial

```
REM simple binding for X! so it's on the
REM main calculator page
```

```
x=Math.Factorial(Calculator.Value)
STOP x
```

#### The Math.IsNaN function

Math.IsNaN(value) returns true (1) when the given value evaluates to a NaN value and 0 otherwise. Object that aren't numbers (or aren't convertible to numbers) will also be a NaN value.

Nan values will propagate their values in BC BASIC, and don't compare equal to each other. You cannot use the check **IF value = Math.NaN THEN PRINT "IS NAN"** because two NaNs are never equal to each other. The only simple way to tell a number is a NaN value is to use the Math.IsNaN function.

To set a variable to NaN, set it to Math.NaN.

### 33.7.6 Math.PI, Math.E and Math.NaN values

Two constants, Math.PI and Math.E are available from the Math extension.

#### Example of using Math.PI and Math.E:

```
REM Converts a circle AREA to DIAMETER
REM area = Math.PI * R**2, which means
value = Calculator.Value
retval = 2 * SQR (value / Math.PI)
Calculator.Message = "Converted area " + value + " to
diameter"
Calculator.Value = retval
```

The Math.NaN value is for “Not a number”. To test a value to see if it’s a NaN, use the Math.IsNaN function.

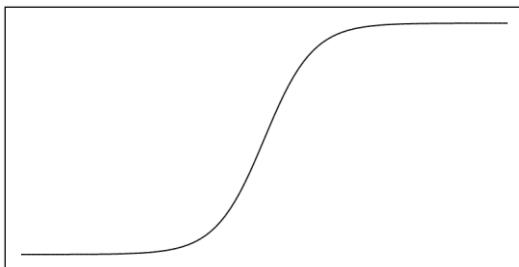
### 33.7.7 Math.Fft and Math.InverseFft Fourier transform

Math.Fft (array) will compute the Fourier transform of the given data. This lets you move from the time domain to the frequency domain; it’s often used when processing samples to remove noise.

Math.InverseFft (array) calculates the inverse Fourier transform

### 33.7.8 Math.Sigma (x)

The Sigma function is used to convert an x value which has any value into one that has a range from 0 to 1.



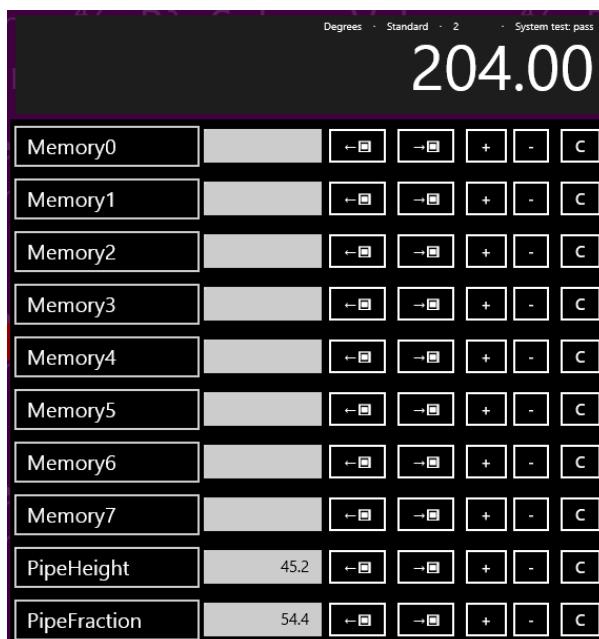
This graph of the Sigma function (showing a range from -20 to +20) was generated by this little mini-program:

```
CLS WHITE BLACK
PRINT "Sigma Function"
g = Screen.Graphics (50, 50, 200, 400)
g.Border = BLACK
```

```
DIM d()  
RANGE = 20  
FOR x = -(RANGE/2) TO (RANGE/2) STEP (RANGE/100)  
    y = Math.Sigma (x)  
    d.AddRow (x, y)  
NEXT x  
g.GraphXY (d)
```

### 33.8 MEMORY EXTENSION

You can read and write string and numeric values to any of the calculator memory. In the example, there are 8 unnamed memory cells (Memory0 to Memory7) plus two named cells (PipeHeight with a value of 45.2 and PipeFraction with a value of 54.4).



Some named memory values are displayed in the Memory screen of Best Calculator. Display the Best Calculator memory screen by tapping the **Memory** menu item.

#### 33.8.1 Memory[<expression>] and Memory.<name>

There are three ways to access a cell: by number, by name, and by simple name.

**Access cells by number:** `Memory[<expression>]`. The named cells can also be accessed by number; in the picture, PipeHeight is the cell right after Memory7 and is accessed as `Memory[8]`.

**Access cells by name:** `Memory[<expression>]`. For example, the PipeHeight cell can be accessed as `Memory["PipeHeight"]`

**Access cells by simple name:** `Memory.<constant_name>` where the constant looks like a variable name (without double quotes) and not a string or number. The memory cell name must be compatible with the rules for variable names. For example, the name can't have spaces or start with a number. The PipeHeight cell can be accessed as `Memory.PipeHeight`.

### 33.8.2 GetOrDefault and IsSet functions

**Is the memory set?** You can tell if a memory cell is set or not in two ways:

`Memory.GetOrDefault(<expression>, <default value>)` returns either the memory value (if it's set) or the supplied default value if not.

`Memory.IsSet(<expression>)` returns true or false (1 or 0) if the memory value is already set.

These functions are commonly used to let you “smart initialize” a value. For example, some calculations use a seldom-changed value (for example, money conversions). You can use `Memory.GetOrDefault` as the default value in an input expression and then save the value that the user enters.

**Example money conversion program:**

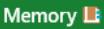
```
REM
REM The defaults here are roughly the conversion
REM rate from yen to australian dollars.
REM 1 yen is about 0.011 australian dollar;
REM 10000 yen is about 110 australian dollars.
REM
prompt1 = "Conversation rate <from> to <to>" ↴
    + "[e.g., yen to australian dollars]"
prompt2 = "Amount to convert [e.g., amount in yen]"
rate = INPUT DEFAULT Memory.GetOrDefault ↴
    ("ConversionRate", 0.011) PROMPT prompt1
Memory.ConversionRate = rate
```

```
amount = INPUT DEFAULT Memory.GetOrDefault ↴
    ("ConversionAmount", 10000) PROMPT prompt2
Memory.ConversionAmount = amount
value = amount * rate
Calculator.Message = "Convert " + amount + ↴
    " at a rate of " + rate + " is " + value
Calculator.Value = value
```

### 33.8.3 Memory technical details

**The calculator memory is both persistent and roaming.** Persistent means that it keeps its value between program runs; it's never automatically reset to zero or other default states. Roaming means that the data roams between your sessions on different computers.

The DUMP command will print out all the memory values to the scrolling console screen.

Best Calculator Memory display will show the first 10 memory slots. However, you can actually access more than that with BC BASIC. Memory slots in BC BASIC can be numbered up to 100. You can show the Best Calculator Memory screen by pressing the Memory  key on the left menu.

Interesting cases for programmers:

1. New in the 2017 release: memory can now be strings as well as doubles!
2. Using indexes less than 0 or more than 100 will silently fail, as will numeric indexes which are not integer (e.g., 3.5). These reads will always return a no such value.
3. An integer index and the string version of the index will refer to the same cell. For example, Memory[1] and Memory["1"] refer to the same memory cell.
4. If you used a named cell and there isn't a cell already with that name, the cell won't be visible in the display. If the user then renames a cell in the memory display with that name, future reads and writes will be to that user-named cell. BC BASIC doesn't place any limit on how many cells there are. Given any particular name, BC BASIC will prefer to save and load from the visible memory cells, but will use the non-visible cells if it has to. BC BASIC won't ever change the name of a cell; that's up to you.

5. There isn't any way to delete memory cells. Once set, the memory cell is created forever. You can, of course, override the memory value.

**Examples of using the Memory extension:**

```
CLS  
  
REM  
REM You can use integer index values  
REM  
Memory[0] = Memory[0] + 1  
Memory[1] = Memory[1] +10  
PRINT "Numeric Index: "; Memory[0]; " "; Memory[1]  
  
REM  
REM You can use simple index names  
REM  
Memory.PipeHeight = Memory.PipeHeight + 1  
PRINT "Simple name: "; Memory.PipeHeight  
  
REM  
REM You can use index names with square brackets  
REM  
Memory["PipeHeight"] = Memory["PipeHeight"] + 1  
PRINT "Const Index: "; Memory["PipeHeight"]  
  
REM  
REM You can use variables and expressions  
REM      in the index name  
REM  
name = "PipeHeight"  
Memory[name] = Memory[name] + 1  
PRINT "Variable Index: "; Memory[name]  
  
prefix = "Pipe"  
suffix = "Height"  
Memory[prefix + suffix] = ↓  
      Memory[prefix + suffix] + 1  
PRINT "Expression Index: "; ↓  
      Memory[prefix + suffix]  
  
REM Some memory isn't set  
a = Memory.NotSet  
isset = Memory.IsSet ("PipeHeight")  
isnotset = Memory.IsSet ("NotSet")  
ns = Memory.NotSet =Memory.NotSet  
  
REM Memory.GetOrDefault returns either the
```

```
REM memory value or the default value  
REM depending on whether the memory was  
REM set or not.  
default = Memory.GetOrDefault ("NotSet", 34)  
notdefault = ↓  
    Memory.GetOrDefault ("PipeHeight", 34)
```

DUMP

### 33.9 SCREEN EXTENSION

The Screen extension gives you additional information about the graphics screen.

#### 33.9.1 Screen.ClearLine(<line>) and Screen.ClearLines(<from>, <to>)

You can clear an entire line on the screen with Screen.ClearLine (<linenumber>). This is very useful with doing data-collection work; you can provide a constantly updated display on the screen with minimal effort

Example:

```
Screen.ClearLine(3)  
PRINT AT 3,1 "Newest Data";X
```

The Screen.ClearLines(from, to) function is similar but it clears an entire set of rows at once.

#### 33.9.2 Screen.RequestActive() and Screen.RequestRelease()

These two powerful functions let you keep the computer screen on even when the user is not interacting with the computer. These are very commonly used in Bluetooth and IOT applications when you need your application to run for an extended period of time without any user activity. Normally a computer or phone screen will be automatically turned off by the operating system after a period of time.

The Screen.RequestActive() will ask for the screen to stay on and Screen.RequestRelease() will allow the screen to turn off. They are counted; if you call Screen.RequestActive() twice you have to call Screen.RequestRelease() twice for the screen to be allowed to turn off.

Important: only use these for long-running applications that must remain active

with not human interaction. Most programs do not fall into this category.

Keeping the screen on will drain your batteries and is only recommended when you know the device will be continuously powered.

### 33.9.3 Screen.H and Screen.W Extension

The Screen.H and Screen.W provide the height and width, in fixed-width characters, of the screen. The most common use is to help lay out text to fix a particular screen size.

**Example: using Screen.H and Screen.W to print in the middle of the screen:**

```
CLS MAGENTA
PrintCenter ("Hello, world!")

FUNCTION PrintCenter (str)
    lmargin = 1+INT ((Screen.W - LEN str) / 2)
    IF (lmargin < 1) THEN lmargin = 1
    row = INT ((Screen.H) / 2)
    PRINT AT row,lmargin str
    RETURN
```

The resulting output shows a neatly-centered message.



### 33.9.4 Screen.GX and Screen.GY

You can now get the graphic size of the terminal screen with the Screen.GX and Screen.GW values. These are useful to make a graphics display that's neatly laid out on the terminal screen.

In the example, text is placed on the left side of the screen and a graphical area is on the right. The graphics position includes the border, but the size does not.

```
REM Make a small text area on the left
REM and a graphics area on the right

CLS WHITE BLACK
PRINT AT 1,1 "*---5----*---5----*---5----*---5"
PRINT AT 2,1 "*-----1-----2-----3-----"
PRINT AT 4,1 "This text area is on the left"
PRINT AT 5,1 "Screen.W", Screen.W
PRINT AT 6,1 "Screen.H", Screen.H
PRINT AT 8,1 "The graphics area is on the right"
PRINT AT 9,1 "Screen.GW", Screen.GW
PRINT AT 10,1 "Screen.GH", Screen.GH
PRINT AT 16,1 "Line 16"

REM The graphics screen is placed based on the
REM outer position of the graphics (including the
REM border area). The screen size is based on
REM the inner area.
REM Hence the need for account for the padding.
PADDING = 1
X1 = Screen.GW/2 - PADDING
Y1 = 0
W = Screen.GW - X1 - PADDING*2
H = Screen.GH - Y1 - PADDING*2
g= Screen.Graphics (X1, Y1, H, W)
g.Border = BLACK
g.Fill = WHITE
g.Circle (W/2, H/2, Math.Min (W/2, H/2)-5)
```

The output looks like this. The screen was resized to have a 16 rows of 60 columns instead of the normal 24x80 size. Earlier version of BC BASIC had a smaller default screen size.

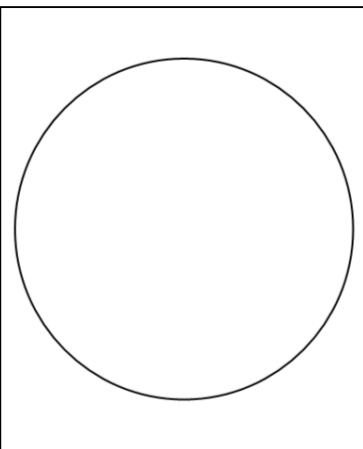
# 16x60 (12)

\*---5---\*---5---\*---5---\*  
\*-----1-----2-----3

This text area is on the left  
Screen.W 60  
Screen.H 16

The graphics area is on the right  
Screen.GW 396  
Screen.GH 239

Line 16



A large circle is centered within a square frame. The square frame has a thin black border and is positioned on the right side of the text area. The circle is also drawn with a thin black line.

The output will always have the same proportions regardless of the characters size of the screen or the size of the font.

## 33.10 SCREEN.GRAPHICS() AND SCREEN.FULLSCREENGRAPHICS()

### EXTENSION

The Screen extension includes a **Graphics()** and **FullScreenGraphics()** method that create a new graphs window on the screen. You can directly draw lines on the graphic screen or you can create an automatic graph that will quickly and easily draw your data and automatically update when the data changes.

You can make a new graphics screen either with a default position and size and set them later (`g = Screen.Graphics()` and then `g.SetPosition(X, Y)` and `g.GetSize(H, W)`) or make a graphics screen at a given location (`g = Screen.Graphics(X, Y)` or make a screen with a starting location and size (`g = Screen.Graphics(X, Y, H, W)`).

You can also make a full-size graphics area with `Screen.FullScreenGraphics()`. This graphics area works just like the one from `Screen.Graphics()` except that it's bigger. Despite the name, the "full screen" only covers the BASIC programming area. There will always be a small "X" in the upper-right corner; tap it to make the screen vanish temporarily. This can help you reclaim control of your program. The full screen graphics area is especially useful when you want to control more aspect of your program's UI. The full screen graphic will vanish when the program is complete.

Once you have a screen graphics object you can draw on it or make automatic graphics on it using the `g.GraphY(data)` or `g.GraphXY(data)` methods. You can also add buttons and sliders to make an interactive program.

#### 33.10.1 Border = color

You can set the color of the border of a graphics window. This only effects graphics objects which are the main window (e.g., created by `Screen.Graphics()`). You can set the color to any of the normal BC BASIC colors.

Example:

```
g = Screen.Graphics(50, 50, 200, 400)
g.Border = BLACK
```

#### 33.10.2 Clear()

The `Clear()` method on a graphics screen will remove all of the objects from the screen.

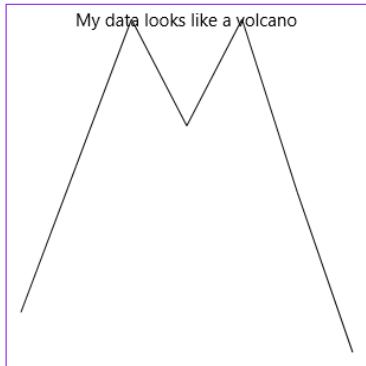
### 33.10.3 GraphXY(data)

You can also make an automatic graph directly from your data. This sample shows how to make a small array, add rows of X,Y data, and then display the data. The graph is given a title.

In this graph,

```
CLS GREEN
PRINT "Display some XY data"
DIM data()
data.AddRow(1,1)
data.AddRow(2,10)
data.AddRow(3,15)
data.AddRow(6,15)
data.AddRow(7,3)
data.AddRow(10,-1)

g = Screen.Graphics()
g.Title = "My data looks like a volcano"
g.GraphXY (data)
```



The GraphY() automatic graph takes an array (from the DIM statement) that contains Y (up-and-down) values. The X value for the graph is taken directly from the index of the data (1, 2, 3, and so on).

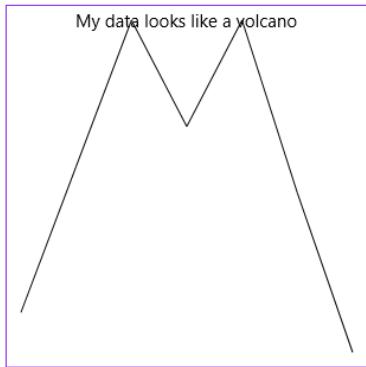
XYGraphs will automatically resize the Y axis to fit the data. You can also force the graph.YAxisMin and graph.YAxisMax value to specific values.

### 33.10.4 GraphY(data, [name])

Automatic graphs can also be created with just the Y data. For these graphs, the X data is simply the index into the data array.

```
CLS GREEN
PRINT "Display some data"
DIM data()
data.Add(1.1)
data.Add(2.2)
data.Add(3.3)
data.Add(2.5)
data.Add(3.3)
data.Add(2.0)
data.Add(.8)

g = Screen.Graphics()
g.Title = "My data looks like a volcano"
g.GraphY (data)
```



The GraphY() automatic graph takes an array (from the DIM statement) that contains Y (up-and-down) values. The X value for the graph is taken directly from the index of the data (1, 2, 3, and so on).

If you provide an (optional) name, you can update the graph at will. For example, if you are getting a period “dump” of data, you can make an automatic graph with a name. Each time you get more data, call g.GraphY(data, “name”) and the named graph will be redrawn with the new data. If the named graph hadn’t already existed, it will be created for you.

### 33.10.5 SetPosition(X,Y) and SetSize(H, W)

You can set the size and position of a graphics window with the `graphics.SetPosition (X, Y)` and `graphics.SetSize (height, width)` methods. This little example creates two windows and draws something different in each.

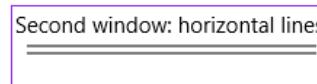
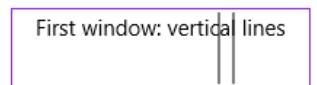
```
CLS
PRINT "Position and resize the graphics windows"

g1 = Screen.Graphics()
g1.Cls()
g1.Title = "First window: vertical lines"
g1.SetPosition (100, 50)
g1.SetSize (50, 200)
g1.Line (140, 1, 140, 50)
g1.Line (150, 1, 150, 50)

g2 = Screen.Graphics()
g2.Cls()
g2.Title = "Second window: horizontal lines"
g2.SetPosition (100, 150)
g2.SetSize (50, 200)
g2.Line (1, 20, 200, 20)
g2.Line (1, 25, 200, 25)
```

The result is two smaller windows that are moved over a little.

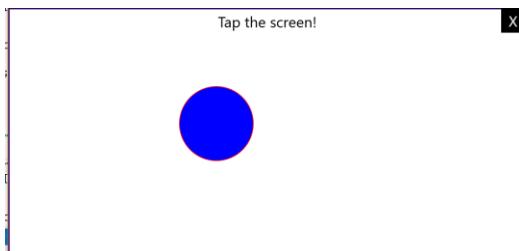
#### Position and resize the graphics windows



### 33.10.6 SetMoved(f, arg), SetPressed(f, arg), SetReleased (f, arg)

The `SetMoved`, `SetPressed` and `SetReleased` functions all work the same way: you pass in the name of a function to call and an argument, and the function will be called when the user presses down on the mouse, moves the mouse, or releases the mouse on the graphic.

The function whose name you give will get four arguments: the graphics object, the X and Y position of the tap, and the arg that was originally passed in.



*Screenshot of the example code running. The user is pressing their finger on a touch-screen in the center of the blue dot. The blue dot tracks their movement.*

Example code. It creates a full screen graphics area, adds in text and a little blue dot (which is set to invisible by setting the Opacity to 0), and then sets up the callbacks. Since I don't use the callback argument value, I don't set it to anything special.

```
g = Screen.FullScreenGraphics()

text= g.Text (g.W/2-150, g.H-75, g.W/2+150, g.H, "Tap
the screen!", 20)
text.Alignment = "CC"
dot = g.Circle (0, 0, 50)
dot.Opacity = 0
dot.Fill = BLUE
dot.Stroke = RED

g.SetMoved ("OnMoved")
g.SetReleased ("OnReleased")
g.SetPressed ("OnPressed")

FOREVER

FUNCTION OnMoved(g, x, y, arg)
    GLOBAL dot
    dot.CX = x
    dot.CY = y
RETURN

FUNCTION OnPressed(g, x, y, arg)
    GLOBAL dot
    dot.Opacity = 1
    dot.CX = x
    dot.CY = y
```

```
RETURN
```

```
FUNCTION OnReleased(g, x, y, arg)
    GLOBAL dot
    dot.Opacity = 0
RETURN
```

### 33.10.7 Updating Data with graph.Update() and PAUSE

When you update the data in the DIM'd array that you passed into GraphY, the graph will automatically update. The update happens when you do a PAUSE or FOREVER command or when you call graph.Update()

In the example, a SIN, COS and TAN window are created and constantly updated. In the FOR loop at the end, the angle variable is slightly incremented on each loop. From the angle variable the SIN COS and TAN values are calculated and added to the corresponding arrays (sinData, cosData and tanData). Because each of these arrays has a MaxValue value (each is 100) and their RemoveAlgorithm is set to "First", when the 101<sup>st</sup> item is Add'ed to the arrays, the whole array is shifted over.

The graphs are redrawn during the PAUSE 1 statement.

```
CLS WHITE BLACK
```

```
PRINT "SIN, COS and TAN updates"
gSin = Screen.Graphics()
gSin.Background = WHITE
gSin.Stroke = BLACK
gSin.SetPosition(100, 50)
gSin.SetSize(75, 200)
gSin.Title = "SIN wave"
```

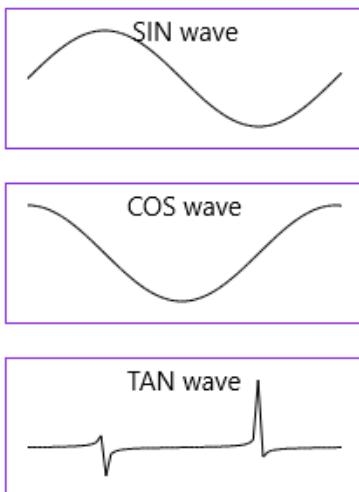
```
DIM sinData()
sinData.MaxCount = 100
sinData.RemoveAlgorithm = "First"
gSin.GraphY(sinData)
```

```
gCos = Screen.Graphics()
gCos.Background = WHITE
gCos.Stroke = BLACK
gCos.SetPosition(100, 150)
gCos.SetSize(75, 200)
gCos.Title = "COS wave"
```

```
DIM cosData()
```

```
cosData.MaxCount = 100  
cosData.RemoveAlgorithm = "First"  
gCos.GraphY(cosData)  
  
gTan = Screen.Graphics()  
gTan.Background = WHITE  
gTan.Stroke = BLACK  
gTan.SetPosition(100, 250)  
gTan.SetSize(75, 200)  
gTan.Title = "TAN wave"  
  
DIM tanData()  
tanData.MaxCount = 100  
tanData.RemoveAlgorithm = "First"  
gTan.GraphY(tanData)  
  
FOR angle = 0 TO 25 STEP .1  
    sinData.Add (SIN(angle))  
    cosData.Add (COS(angle))  
    tanData.Add (TAN(angle))  
    PAUSE 1  
NEXT angle
```

## SIN, COS and TAN updates



The resulting three graphs during the middle of the program run.

### 33.11 SCREEN.GRAPHICS OBJECTS (ARC, BUTTON, SLIDER, TEXT AND MORE)

The point of creating a graphics display area with Screen.Graphics() and Screen.FullScreenGraphics() is to add graphical objects. Some objects like the Arc(), Circle(), Polygon() and Text() are static objects; you can set them and change them, but the user won't directly manipulate them. Other objects like the Slider() and Button() are interactive object; a Button can be pressed and the Slider can be moved. Naturally you can connect a button press to your own BASIC function.

#### 33.11.1 Common fields X1 Y1 X2 Y2 CX CY Opacity Rotate Data

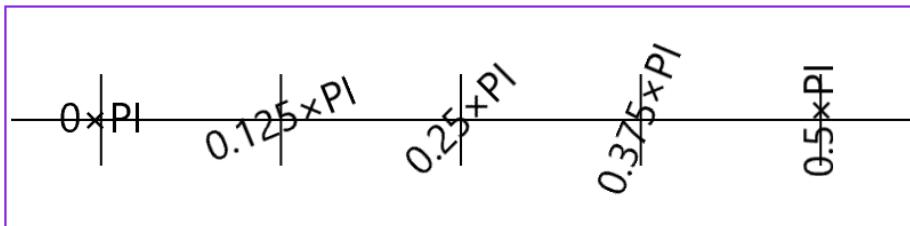
The graphic objects have some fields in common. Notably, most objects will let you set the **X1**, **Y1**, **CX**, **CY**, **X2** and **Y2** values (for the X1,Y1 and X2,Y2 position and the center CX and CY values). You can also read these values back.

BC BASIC treats the X1,Y1 and X2,Y2 points differently: when you set the X1 and Y1 points, the object moves but doesn't change size; the X2 and Y2 are automatically updated to maintain the size. When you change the X2 and Y2 positions, the size will be updated.

You can set the **Opacity** of most graphical objects. An opacity of 0 means that the object is invisible and an opacity of 1 means the object is fully visible and that objects behind the object are not visible.

You can set the **Rotate** value on most graphical objects. The value is normally 0; positive values rotate counter-clockwise using an amount in radians. You can adjust exactly where the rotation happens with the CXD and CYD (center x/y delta) values; these are normally .5 to indicate that the rotation is around the center of the object.

Demonstrate rotating text  
by setting `text.Rotate=value_in_radians`



This example program displays text with different rotate amounts,

```
CLS WHITE BLACK
PRINT "Demonstrate rotating text"
PRINT "by setting text.Rotate=value_in_radians"

g = Screen.Graphics (0, 50, 100, 400)
w = g.W / 5
NCOL = 4
y = g.H/2
g.Line (0, y, g.W, y)
FOR c = 0 TO 4
    ang = Math.Round (c*PI/(2*NCOL), 3)
    angText = "" + Math.Round (ang/PI, 3) + "xPI"
    x = c * w + w/2
    t = g.Text (x-40, y-14, x+40, y+14, angText, 18)
    t.CXD = .5
    t.CYD = .5
    t.Alignment = "CC"
    t.Rotate = ang
    g.Line (x, y-20, x, y+20)
NEXT c
```

Most objects include a **Data** value to store data. The objects will simply preserve the Data field and will not use it or change it. Games, for example, can use this field to store the “damage” that running into an object will cause.

### 33.11.2 Common methods: obj1.Intersect(obj2)

You can tell if two graphical objects intersect using the **obj1.Intersect(obj2)** method. It will return 1.0 (TRUE) if the two object overlap at all, and 0.0 (FALSE) if they do not. This is especially useful in games. For example, if you make a PONG type game you can see if the ball has intersected the wall or the paddles.

The intersection is done based on the overall “bounding box” of the object. Two circles, for example, might not visibly overlap and might still be considered to intersect. Additionally, the Rotate value is not taken into account at all.

You can mix and match the graphical objects for the Intersection testing. You can only test objects which are “rectangular” objects in that they have a definite x1, y1, x2, y2 location. (The Line method does not make an object).

### 33.11.3 Arc (cx, cy, innerRadius, outerRadius, ang1, ang2)

Draws an arc centered around point [cx,cy] starting at angle ang1 and rotating through to ang2 (both specified in radians). The arc is drawn “thick” with an inner and outer radius. The zero point for ang1 and ang2 is to the right of the [cx,cy] center point.

### 33.11.4 Button (x1, y1, x2, y2, “text”, “function”)

You can add a button (  ) to a graphics area with the graphics.Button function. You have to specify an actual graphics screen (see the example under slider to see how). The size and location of the button is determined by the button’s x1, y1, x2, y2 values.

The “text” value is the text to display in the button, and “function” is the name of a function to call when the button is clicked.

The position can be changed programmatically: **button.X1 = 100** will move a button to X=100. If you change just the X1 value, both the X1 and X2 values will change so that the overall width stays the same. But you can change just the X2 value and the X1 value will not change. Similarly, changing the Y1 value will change both the Y1 and Y2 values, keeping the height the same.

### 33.11.5 Circle(X, Y, radius [,yradius]) Line(X1, Y1, X2, Y2) Rectangle(X1, Y1, X2, Y2)

A quick sample that draws circle, a line, and a rectangle (both filled and unfilled)

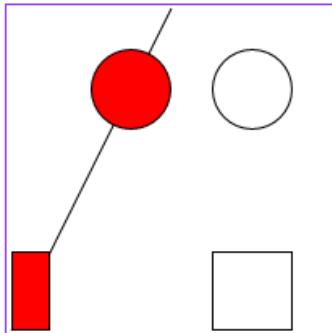
```
CLS WHITE BLACK
PRINT "Circles, Lines, Rectangles"
g = Screen.Graphics()
g.SetSize (200, 200)
g.SetPosition (100, 100)
g.Background = WHITE
g.Stroke = BLACK
g.Fill = RED
g.Line (1, 1, 100, 200)

g.Rectangle (1, 1, 25, 50)

g.Circle (75, 150, 25)

g.Fill = NONE
g.Rectangle (125,1, 175, 50)
g.Circle (150, 150, 25)
```

## Circles, Lines, Rectangles



The Rectangle objects also includes a Data member that you can set.

The program first creates a graphics windows (`g = Screen.Graphics()`) and then draws a line, a filled rectangle and circle and a non-filled rectangle and circle.

If you supply three parameters to the Circle method, the values are the x and y centers and the radius of the circle. If you pass in four paramters, the values are the x and y centers, the x radius and the y radius. This lets you make an ellipse.

You can call the method “**Ellipse**” instead of “Circle”.

The Rectangle and Circle (and Ellipse) methods will return an object that can be modified. For example, the flames in the Happy Birthday sample program are ellipses whose size changes every time a note is played. This produces an interesting flicker effect.

```
REM Make a circle, and then make
REM it wider
CLS GREEN
PRINT "Change an object"
g = Screen.Graphics(50, 50, 200, 200)
circle = g.Circle(40, 40, 10)
REM Make the circle wider
circle.X1 = 40-10-10
circle.X2 = 40+10+10
```

You can also set the **Stroke** and **Fill** of the object.

The radius of the circle can be set later on. This is useful when animating objects; by making them larger, you can draw attention to the circle.

```
circle = g.Circle (100, 100, 30)
circle.Radius = 40
```

### 33.11.6 LineTo (x, y), MoveTo (x, y), and ClearGoTo()

Often you want to draw a series of lines which abut each other (like in a triangle). You could use the Line (x1, y1, x2, y2) method three times, repeating each point, or you could do a MoveTo (x1, y1), LineTo (x2, y2), LineTo (x3, y3) and a last LineTo (x1, y1).

Each of the regular graphics command will update the “current position”.

To clear the current position, use ClearGoTo().

### 33.11.7 Polygon() and poly.AddPoints() and SetPoints ()

Creates a polygon with any number of sides and set or add the vertices.

Example: make a triangle (from the Circle, Line and Rectangle demo program in EX: Graphics)

```
tri = g.Polygon()
tri.SetPoints (70, 80, 95, 130, 120, 80)
```

You can also add to the number of points in the polygon with AddPoints (x1, y1, ...). Like SetPoints, you can add any number of additional points. The AddPoints and SetPoints methods can each take in an array; the array needs to just a list of points (x1, y1, x2, y2, ...) and not a 2-D array.

Longer example: how to draw a star with arbitrary number of points:

```
REM Specify the x,y position, the number of points
REM the inner and outer size, and the ang.
REM If you don't pick an angle, PI/4 is used so that there
REM is a point at the top.
FUNCTION DrawStar (g, x, y, npoints, inner, outer, ang)
  IF (Math.IsNaN(ang)) ang = PI/4
  poly = g.Polygon()
  delta = (2 * PI / npoints)
  FOR I = 1 TO npoints
```

```
a1 = (I-1) * delta + ang  
a2 = a1 + delta / 2  
x1 = x + COS(a1)*outer  
y1 = y + SIN(a1)*outer  
x2 = x + COS(a2)*inner  
y2 = y + SIN(a2)*inner  
poly.AddPoints (x1, y1, x2, y2)
```

NEXT I

RETURN

When called repeatedly you can get results like this selection of random stars



### 33.11.8 Slider (x1, y1, x2, y2, "text", "function")

You can add a slider to a graphics area with the Screen.Slider function. You have to specify an actual graphics screen (see the example to see how). The size and location of the slider is determined by the button's x1, y1, x2, y2 values. If the overall size is wider than it is high, the slider will be a horizontal slider; otherwise it will be a vertical (up and down) slider.

The "text" value is the text to display as a header, and "function" is the name of a function to call when the slider is changed. The function is called with the slider and the new value.

The default values for the slide range from 0 to 255, and the starting position is 0. You can change the slider.Min and slider.Max value to set the min and max values. You can also set the Text to display and the Value amounts.

You can get the Max, Min, Text and Value amounts as well.

The position can be changed programmatically: `slider.X1 = 100` will move the slider to X=100. If you change just the X1 value, both the X1 and X2 values will

change so that the overall width stays the same. But you can change just the X2 value and the X1 value will not change. Similarly, changing the Y1 value will change both the Y1 and Y2 values, keeping the height the same.

**Example:** draw a button, a horizontal slider and a vertical slider

```
CLS BLACK WHITE
PRINT "Sliders and Button Demo"
g = Screen.Graphics (50, 70,300, 200)
g.Background = GRAY
hslider = g.Slider (0, 0, 200, 100, ↓
    "My Horizontal", "OnHSliderChange")
vslider = g.Slider (150, 100, 200, 300, ↓
    "Verti.", "OnVSliderChange")
button = g.Button (0, 200, 90, 230, ↓
    "Button", "OnButtonClick")
FOREVER

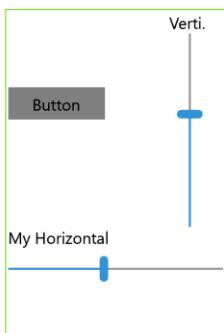
FUNCTION OnButtonClick(button)
    Screen.ClearLine (4)
    PRINT "Button clicked!"
    FOREVER STOP
RETURN

FUNCTION OnHSliderChange(slider, value)
    Screen.ClearLine (2)
    PRINT "HSlider", value
RETURN

FUNCTION OnVSliderChange(slider, value)
    Screen.ClearLine (3)
    PRINT "VSlider", value
RETURN
```

Results

Sliders and Button Demo  
HSlider 113  
VSlider 150



### 33.11.9 Text (x1, y1, x2, y2, "text", fontsize)

Add text to your graphics area with the `graphics.Text()` method. Like the `Button` and `Rectangle` methods, the `Text` method defines a rectangular area on the screen with an `x1, y1, x2, y2` positions. The next two arguments are for the text to display and the font size.

Once you have a text object, you can set the **Text** value, the **Alignment** and the **FontSize**.

The **Alignment** is a string that must be two characters long. The first sets the left/right alignment and can be one of L C R or S. These stand for Left, Center, Right and Stretch. The second character sets the up/down alignment and can be one of T C B or S which stand for Top, Center, Bottom and Stretch.

For example, `mytext.Alignment = "LT"` will set the text alignment to the left and top.

When you set the `Text`, any `\n` in the string will be replaced with a carriage-return (CR) character. This lets you make multi-line output.

## 33.12 SCREEN.GRAPHICS SCALING (ADVANCED)

BC BASIC includes three methods on the `graphics` object to help scale the graphical output. These methods are `g.SetScaleWindow (name, x1, y1, x2, y2)` to make a window, `g.SetScale (name, x1, y1, x2, y2)` to define the scaling in physical units, and `g.UseScale (name)` to use a particular scaling.

So far, the graphics you've seen involves carefully placing the lines and circles at whatnot onto the screen so that they would be visible. But sometimes you want to use some natural unit, and draw your graphics in those units. For example, you might be creating a simulation of the solar system, and want to draw everything in terms of AU (astronomical units).

BC BASIC can automatically convert your preferred units into the graphics space. This is called "scaling". In the screenshot of the "bubble simulation" program, the bubble physics are done in the cgs

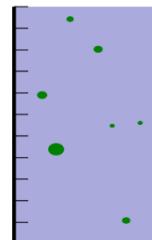
(centimeter/grams/seconds) system, where (for example) the acceleration of gravity is 980 cm/second<sup>2</sup>. The beaker is 50cm wide and 100 cm tall.

To set up a scaling area, you have to do three things:

1. Create a "window" on the screen in graphics co-ordinates and give it a name. This is where the scaling area will be created. In the bubble example, this is  
`g.setScaleWindow ("bubble", xpos, 0, xpos+w, h)` where xpos is  $\frac{1}{2}$  of the width of the screen so that the scaling area is on the right hand side of the screen. The W is the rest of the screen, but "squeezed" so that beaker's proportions are neatly scaled.
2. Define the "scale size" scaling to be done in the window. You specify the name of the window and the x and y values of the lower-left and upper right corners of the "window". You specify these in the "physics" units. In the case of the bubbles, the lower left corner of the "window" is 0,-10 (x=0, and y=-10 so that the beaker will "hover" in the air a little). The upper-right corner is x=width of the beaker and y=20% more than the height of the beaker so that the bubbles have some place to go.  
`g.setScale ("bubble", 0, -10, BeakerW, BeakerH*1.2)`
3. Use the "window" by name with the `g.UseScale ("name")` method.

Bubble Animation

Small Bubble Animation  
 Most recent bubble  
 Size=0.06 cm  
 Speed=2.1 cm/s



There's always a "pixel" map which is the standard pixel-type scaling. Setting the scaling only affects future graphical items; setting a new scaling or changing a scaling map doesn't change what's been drawn on the screen.

### 33.13 SENSOR EXTENSION

The Sensor extension (available only in the IOT Edition) lets you access a number of sensors available on different Windows machines. Sensors include compass and inclinometer, light, location and the microphone.

#### 33.13.1 Using Sensor.Camera and making images

This is an experimental feature. You can expect that the API will change or be removed in future versions.

Best Calculator is able to grab images from the built-in camera (front or back) and display the results directly, display a filtered version, and call a callback function with a color analysis of the image.

#### 33.13.2 The simplest camera program

The most basic program lets you display the camera. To do this you need to

1. Make an *image* on a graphics screen. You also have to make the graphics screen.
2. Ask for the sensor camera
3. Tell the camera to display onto the image
4. Tell the camera to start

```
REM Very simple camera program
CLS BLUE
PRINT "Simple camera program"

REM graphic size is x, y, h, w
g = Screen.Graphics (50, 50, 400, 600)
REM image size is x1, y1, x2, y2
img = g.Image (0, 0, 600, 400)

REM Cameras include front, rear, face
camera = Sensor.Camera("front")
camera.Image = img
camera.Start()

FOREVER
```

The resulting camera view uses hardware acceleration to display a high-quality image. This image cannot be analyzed or modified.

### 33.13.3 Using Analyze to modify the Sensor.Camera image

You can make a separate image from the preview, modifying the result. This is done by making an analysis object and setting its properties. One of the most important properties is the View property. This is an image (from a graphics.Image(x1, y1, x2, y2) method call) that the analysis results will be displayed on.

The Analysis can map the incoming red, green and blue values from their original values (from 0 to 255) into a new set of values based on a set of points. This is done with a series of analysis.AddPoint (“<channel>”, input, output). The channel is “r”, “g” or “b”

The default points are (0,0) and (255,255). This is the “don’t make any changes” points.

To turn off one of the channels (a channel is one of the red, green, and blue values), make a map that is (0,0) and (255,0). This mapping will turn any input value (0 to 255) to 0.

To make a “posterized” channel, add in a series of points which are “flat”. For example, to make a three-channel posterized channel where each pixel is either 0, 128 or 255, use these set of points: (0,0) (85,0) (86,128)  
(170,128)(171,255)(255,255)

Each value from 0 to 85 will map to 0; each point from 86 to 170 will be 128, and each value higher will be 255.

In the simple “red” example, only the red (“R”) values are sent through to the final image. This is done

```
REM Make an image to display the analyzed  
REM image on  
analysisg = g.Image (0, 200, 300, 400)
```

```
REM Make a camera analysis that wil  
REM pass the red values through but  
REM remove the green and blue.  
analysis = camera.Analyze ()  
analysis.Image = analysisg  
analysis.AddPoint ("r", 0,0, 255, 255)  
analysis.AddPoint ("g", 0,0, 255, 0)  
analysis.AddPoint ("b", 0,0, 255, 0)
```

### 33.13.4 Example: HTML color from Sensor.Camera

In this example, we display the rear camera, grab an analysis from the central portion, get the average HTML color (in the standard #rrggbb format) and use that color to change the background of the graphics screen.

```
CLS BLUE
PRINT "Show HTML Color"

REM Step 1: make the camera and preview
cam = Sensor.Camera ("front")
grf = Screen.Graphics (50, 50, 340, 440)
view = grf.Image (20, 20, 420, 170)
cam.Start()
cam.Image = view

REM Step 2: make the analysis image
REM and set up callback
a = cam.Analyze()
REM Radius, CX and CY
a.Radius = 0.25
a.CX = 0.5
a.CY = 0.5
a.AnalysisW = 64
a.AnalysisH = 64
viewanalysis = grf.Image(20, 190, 420, 320)
a.Image= viewanalysis
a.Function = "HtmlColor"

FOREVER

REM Step three: HTML color output
REM and set the background color
FUNCTION HtmlColor (r, g, b)
    color = String.Escape ("color", r, g, b)
    REM Output is e.g. #112233
    Screen.ClearLine (2)
    PRINT color
    GLOBAL grf
    grf.Background = color
RETURN
```

### 33.13.5 Sensor.Compass (function)

The compass sensor callback returns three values: the direction to magnetic north, the heading accuracy, and the direction to true north. Many systems do not report true north. All angles are reported in degrees.

See the full example below. Not all systems include this sensor, and it's only available in the IOT Edition.

### 33.13.6 Sensor.Inclinometer(function)

The inclinometer callback reports on the pitch, roll and yaw and the yaw accuracy. All values are in degrees.

See the full example below. Not all systems include this sensor, and it's only available in the IOT Edition.

### 33.13.7 Sensor.Light(function)

The light sensor will report the light intensity level in lux (lx). From Wikipedia, common lux values include

Illuminance (lux)	Surfaces illuminated by
0.0001	Moonless, overcast night sky ( <a href="#">starlight</a> ) <sup>[3]</sup>
0.002	Moonless clear night sky with <a href="#">airglow</a> <sup>[3]</sup>
0.05–0.36	Full moon on a clear night <sup>[4]</sup>
3.4	Dark limit of civil <a href="#">twilight</a> under a clear sky <sup>[5]</sup>
20–50	Public areas with dark surroundings <sup>[6]</sup>
50	Family living room lights (Australia, 1998) <sup>[7]</sup>
80	Office building hallway/ <a href="#">toilet</a> lighting <sup>[8][9]</sup>
100	Very dark overcast day <sup>[3]</sup>
320–500	Office lighting <sup>[7][10][11][12]</sup>
400	<a href="#">Sunrise</a> or <a href="#">sunset</a> on a clear day.
1000	Overcast day; <sup>[3]</sup> typical <a href="#">TV studio</a> lighting
10,000–25,000	Full <a href="#">daylight</a> (not direct sun) <sup>[3]</sup>
32,000–100,000	Direct <a href="#">sunlight</a>

Not all systems include this sensor, and it's only available in the IOT Edition.

### 33.13.8 Sensor.Location Extension

The Location input is only available for Best Calculator, IOT Edition.

To get location data, use the statement `loc = Sensor.Location()` to get a `location` object and then call `loc.Start("functionName")` to start the location data running.

Call `loc.Stop()` to stop location updates.

Location data is given as latitude and longitude; also included is the altitude (which only some systems can provide) and the location accuracy.

```
CLS BLUE
PRINT "Location data"
DIM position()
position.MaxCount = 100
position.RemoveAlgorithm = "First"

g = Screen.Graphics (50, 150, 400, 400)
g.GraphXY (position)
loc = Sensor.Location()
loc.Start("location")
MAXTIME = 100
FOR time = 1 TO MAXTIME
    PRINT AT 3,1 time
    PAUSE 10
NEXT time

FUNCTION location(latitude, longitude, altitude, accuracy)
    GLOBAL position
    position.AddRow (latitude, longitude)
    Screen.ClearLine (4)
    PRINT latitude, longitude, accuracy
RETURN
```

### 33.13.9 Sensor.Microphone Extension

The Microphone is only available in Best Calculator, IOT Edition.

To get data from the microphone, you need to make a microphone object using the statement `mymic = Sensor.Microphone()`. You then start the microphone input with `mymic.Start("myFunction", seconds)`. Your BASIC `myFunction` will be called every "seconds" seconds with an array full of data. A common value for

“seconds” is 1 (meaning you’ll be called every second) and 0.2 (5 times a second).

Stop the microphone input with `mymic.Stop()`.

Use `mymic.Name` to get the operating system name of the microphone.

Here’s a simple program that displays the microphone volume as a graph. The Lumia 650 is fast enough to create displays of the volume.

```
CLS BLUE
PRINT "Microphone data volume"
DIM volume()
volume.MaxCount = 100
volume.RemoveAlgorithm = "First"
maxVolume = 0

g = Screen.Graphics (50, 150, 150, 600)
g.GraphY (volume)
mic = Sensor.Microphone()
mic.Start("microphone")
MAXTIME = 100
FOR time = 1 TO MAXTIME
    PRINT AT 3,1 time
    PAUSE 10
NEXT time

FUNCTION microphone (data)
    GLOBAL volume
    v = data.SumOfSquares / data.Count
    v = SQR(v)
    volume.Add (v)
    Screen.ClearLine (4)
    PRINT "Volume", v
    GLOBAL maxVolume
    maxVolume = Math.Max(maxVolume, v)
    Screen.ClearLine (5)
    PRINT "Max", maxVolume
RETURN
```

### 33.13.10 Full Sensor example

This example shows how to connect to all of the sensors offered in the IOT edition. Each sensor has a similar pattern: first call `<type>sensor = Sensor.<type>("my<type>function")` to start listening. Then make a

**my<type>function** function to get callbacks. Note that the callbacks will be throttled: if too many are sent at once, only the last callback will be delivered. In the example, a timer is fired periodically; after MAXTIME number of seconds, the program will automatically stop.

```
CLS GREEN
PRINT "All sensors"
PRINT "(not all computers have these)"

maxVolume = 0

REM MAXTIME is the max time to run
MAXTIME = 30

startTime = DateTime.GetNow()
compass = Sensor.Compass("Compass")
inclin = Sensor.Inclinometer("Inclinometer")
light = Sensor.Light("Light")
location = Sensor.Location("Location")
mic = Sensor.Microphone("Microphone")

System.Trace (1)
System.SetIntInterval ("Timer", 2000, "")

FOREVER

PRINT AT 12,1 "All done!"

FUNCTION Compass (direction, accuracy, ↴
    trueNorthHeading)
    Screen.ClearLine (5)
    PRINT "Heading", ↴
        Math.Round(direction), ↴
        accuracy, ↴
        Math.Round (trueNorthHeading)
RETURN

FUNCTION Inclinometer (pitch, roll, yaw, accuracy)
    Screen.ClearLine (6)
    PRINT "Inclin.", ↴
        Math.Round(roll), ↴
        Math.Round(pitch), ↴
        Math.Round(yaw)
RETURN

FUNCTION Light (lux)
    Screen.ClearLine (7)
```

```
PRINT "Light", lux  
RETURN
```

```
FUNCTION Location(latitude, longitude, altitude,  
accuracy)  
    Screen.ClearLine (8)  
    PRINT "Position", ↓  
    Math.Round(latitude, 2), ↓  
    Math.Round(longitude, 2), ↓  
    accuracy  
RETURN
```

```
FUNCTION Microphone (data)  
    v = data.SumOfSquares / data.Count  
    v = SQR(v)  
    Screen.ClearLine (9)  
    PRINT "Volume", Math.Round (v, 4)  
    GLOBAL maxVolume  
    maxVolume = Math.Max(maxVolume, v)  
    Screen.ClearLine (10)  
    PRINT "Max Vol.", ↓  
    Math.Round (maxVolume, 4)  
RETURN
```

REM Sets us up to wait for 10 seconds  
REM and then call FOREVER STOP to  
REM stop the 'FOREVER' statement.

```
FUNCTION Timer (arg)  
    curr = DateTime.GetNow()  
    GLOBAL startTime  
    delta = curr.Subtract (startTime)  
    Screen.ClearLine (4)  
    PRINT "TIME", curr.Time,  
  
    GLOBAL MAXTIME  
    IF (delta > MAXTIME) THEN ↓  
        FOREVER STOP  
RETURN
```

### 33.14 STRING EXTENSION

The String extension gives you more ways to parse string (for example, to parse a string from JSON into an array) and to escape string (for example, to prepare a string to be written to a CSV file).

### 33.14.1 String.Escape ("color", red, green, blue)

String.Escape ("color", r, b, b) make a string #RRGGBB. The values r, g and b must be 0 to 255. Values less than zero are turned into zero and values greater than 255 are turned into 255.

### 33.14.2 String.Escape ("csv", <string or array>)

Call String.Escape ("csv", <string>) to convert a string into something that can be written to a CSV file.

CSV (Comma-separate value) files are commonly used to create spreadsheet-like files of data. BC BASIC follows the conventions of [RFC 4180](#) for CSV data.

The escape is most commonly used with an array. An array (made with a statement like **DIM data()**) will be converted into a single line of a CSV file.

```
CLS BLUE
file = File.WritePicker ("Sample Data file", ".csv",
"data.csv")
DIM data()

REM
REM Make a data file line by line
REM
data[1] = "Time"
data[2] = "Data"
line = String.Escape ("csv", data)
file.WriteAllText (line)

data[1] = "8:05"
data[2] = 1.1
line = String.Escape ("csv", data)
file.WriteAllText (line)

data[1] = "8:10"
data[2] = 1.2
line = String.Escape ("csv", data)
file.WriteAllText (line)
```

Note that the File.WritePicker is only available in the IOT edition, not the regular edition of Best Calculator.

After running this program you will have a CSV file that can be read in by Excel:

The screenshot shows a software interface titled "Best Calculator". At the top, there's a toolbar with a dropdown for "A1", a "Time" button, and a "fx" button. Below the toolbar is a table with columns labeled A, B, C, and D. Row 1 contains "Time" in column A and "Data" in column B. Rows 2 and 3 contain data: row 2 has "8:05" in column B and "1.1" in column C; row 3 has "8:10" in column B and "1.2" in column C. Rows 4 through 7 are empty. At the bottom of the interface, there's a toolbar with buttons for "data", a plus sign, and other functions, along with a zoom level of 145%.

	A	B	C	D
1	Time	Data		
2		8:05	1.1	
3		8:10	1.2	
4				
5				
6				
7				

If you call `String.Escape("csv", "string")`, the result is single cell of a CSV file instead of an entire row. The cell will be properly escape and enclosed in double-quotes as needed.

### 33.14.3 String.Escape ("json", <string or array>)

Call `String.Escape ("json", <string>)` to convert a string into something that can be written to a JSON file.

JSON (JavaScript Object Notation) files are commonly used to send and receive data from the internet. BC BASIC follows [RFC 7159](#) conventions for JSON data.

If you provide an array, you'll get back a complete JSON description of your data. Example:

```
DIM list()
list.AddRow("data", 12.34)
list.AddRow("sensor", "ambient")
list.AddRow("index", 33)
json = String.Escape("json", list)
PRINT json
```

The result will be JSON data like this:

```
{  
  "data":12.34,  
}
```

```
"sensor":"ambient",
"index":33
}
```

### 33.14.4 String.Parse("csv", <data string>)

Use `String.Parse("csv", <data string>)` to convert CSV-encoded file data into a data array. The result is always an array-of-arrays, even if the original is just a single line of data (like what you might get from a stream of data from an Arduino device)

For example, suppose you have a CSV file like this:

```
time, data
8:05, 1.1
8:10, 1.2
8:20, 1.4
```

You can read in the data (using the `file = File.ReadPicker(".csv")` method to get the file and then `allText = file.ReadAll()` to get all the data). Then call `data = String.Parse("csv", allText)` and you'll have an array-of-arrays of all your data.

The individual data values will either be strings or doubles. Any value which can be converted to a double, will be converted. Otherwise, the value will be a string value.

You can print the header values like this:

```
header = data(1)
PRINT "header", header(1), header(2)
```

the result is that the words time and data are printed out.

Full code example:

```
CLS BLUE
file = File.ReadPicker ("csv")
IF (file.IsError)
    REM file will contain an error string
    PRINT "ERROR", file
    STOP
END IF
```

```
REM Read the file and convert to an array  
alltext = file.ReadAllText()  
csv = String.Parse ("csv", alltext)  
header = csv[1]
```

```
REM Print the data  
PRINT "HEADER", header(1), header(2)  
FOR index = 2 TO csv.Count  
    data = csv(index)  
    PRINT index-1, data(1), data(2)  
NEXT index
```

### 33.14.5 String.Parse (“json”, <data string>)

The String.Parse (“json”, <data string>) method converts data in JSON format (often downloaded from the internet using the Http.Get() method) into an array. The array will be in object format, meaning that you can pull data out of the array by name.

A JSON string might look something like this:

```
{  
    "by" : "dhouston",  
    "descendants" : 71,  
    "id" : 8863,  
    "kids" : [ 8952, 9224, ...],  
    "score" : 111,  
    "time" : 1175714200,  
    "title" : "My YC app: Dropbox – Throw away your ... ",  
    "type" : "story",  
    "url" :  
        "http://www.getdropbox.com/u/2/screencast.html"  
}
```

Convert the string to an array with `data = String.Parse (“json”, str)`

You can then get the individual data elements

```
PRINT data.by  
PRINT data.id
```

To see if the returned array has a value, use the `data.HasKey(“<name>”)` call.

For example, `data.HasKey(“by”)` will be 1.

**33.14.6 String.Pos (str, lookFor, startingIndex)**

Finds the index position of the lookFor argument in the input str. Only matches starting from the startingIndex will be returned. If there's no match, 0 is returned. For example

```
PRINT String.Pos ("abcdef", "c", 1)
```

A 3 will be printed. The string matches is case-sensitive; in the example “c” will be found but “C” would not be.

**33.14.7 String.Replace (string, replacement, startingIndex)**

Splices the replacement string into the input string at location startingIndex. A total of length characters will be removed from the input string. For example:

```
PRINT String.Replace ("abcdefghijklm", "123", 4)
```

The string abc123ghi will be printed.

**33.14.8 String.ToUpper(str) and String.ToLower(str)**

The String.ToUpper(str) method returns the original string but converted to upper case. For example

```
str = String.ToUpper("Apple")
PRINT str
```

will print “APPLE”. Similarly,

```
str = String.ToLower("Apple")
PRINT str
```

will print “apple”. This is often used to convert user input into a more regular input for the computer to deal with.

## 33.15 SYSTEM EXTENSION

### 33.15.1 System.Errors

The System.Errors value is the most recent set of errors encountered while running programs. It persists from one program run to the next (but doesn't persist after you shut down Best Calculator). This can be useful when debugging Bluetooth issues.

### 33.15.2 System.FolderBasic()

The FolderBasic method will return the operating-system folder where BASIC program are automatically saved. This can be useful when you need to find where everything gets saved!

### 33.15.3 System.FolderTemporary()

The FolderTemporary returns the folder where temporary files are saved.

### 33.15.4 System.SetInterval (function, interval, argument)

The SetInterval method will start calling functionName at the specified interval (in milliseconds), passing in argument.

In this example, System.SetInterval() sets up a callback on the "showtime" function. That function displays the time. When enough time has passed (5 seconds), it also calls FOREVER STOP which breaks out of the FOREVER statement.

```
CLS GREEN
start = DateTime.GetNow()
System.SetInterval ("showtime", 500, "arg")

PRINT "About to FOREVER"
FOREVER
PRINT "All done!"

FUNCTION showtime(arg)
Screen.ClearLine (10)
time = DateTime.GetNow()
PRINT "The current time is", time.Time

REM Stop after 5 seconds
GLOBAL start
delta = time.Subtract (start)

IF (delta > 5) THEN FOREVER STOP
RETURN
```

### 33.15.5 System.Trace (0=off 1=normal)

When you call System.Trace(1), the BC BASIC will start to print trace information into the CONSOLE output. The trace information starts off printing about once per second and will report on the number of statements processed, the most recent statement processed, the number of callbacks (like from Bluetooth IOT devices) that have been processed and the number that have been suppressed.

Call System.Trace(0) to turn tracing back off.

Tracing persists from one program run to the next. In the sample is a TRON and TROFF program that will. Tracing will stay on until Best Calculator is existed and re-run (or until System.Trace(0) is called)

### 33.15.6 System.Version

The System.Version value lets you see the current version and edition of Best Calculator

Example

```
PRINT System.Version
```

Which might print out something like

Best Calculator 3.8 (X64)

The (X64) shows that the system architecture is a standard X64 Intel-type computer.

## 34 BLUETOOTH PROGRAMMING WITH BEST CALCULATOR, IOT EDITION

---

Best Calculator, IOT edition is a special version of Best Calculator that lets you control multiple IOT devices using Bluetooth via the built-in BASIC programming language. You should have a basic understanding of Bluetooth devices to use these features.

### 34.1 PROGRAMMING BLUETOOTH USING BC BASIC

Let's start with a simple example: let's display the names of each *paired* Bluetooth device

```
CLS BLUE
PRINT "Bluetooth Functions"

REM
REM How many Bluetooth devices are available?
REM
devices = Bluetooth.Devices ()
PRINT "Count", devices.Count

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    PrintBluetoothInfo (bt)
NEXT i

FUNCTION PrintBluetoothInfo(bt)
    PRINT "NAME", bt.Name
    PRINT "ID", bt.Id
    PRINT "PROPERTIES", bt.Properties
    PRINT ""
RETURN
```

The **devices = Bluetooth.Devices ()** line gets all the paired Bluetooth LE devices and puts the results into an array. Then the code simply loops through the devices. We get each device and call **PrintBluetoothInfo**, passing in the device. Each device has a **Name** property with the Windows name of the device. We can also print out the device Id and the Properties.

The `Bluetooth.Devices()` method does not return every possible Bluetooth device. Devices it does not return include

- Devices which the user has not paired. BC BASIC does not include any functions to help the user pair their device
- Devices which are reserved for the system. This includes HID device like mice and keyboards, and audio devices like speakers and headphones
- Bluetooth beacons like the popular Bluetooth enabled luggage tags or the Bluetooth beacons found in some stores, museums, and work places.

Although Best Calculator, IOT edition gives you access to many of the Windows 10 Bluetooth capabilities, there are still areas where you might need to add your own customizations. The IOT edition of Best Calculator comes with the full source code for the calculator. Advanced programmers will find that they can add additional capabilities in C#.

In this and the next chapters, you'll learn

- An introduction to programming Bluetooth using BC BASIC
- The different stages of initializing and programming your device
- The different *objects* that used for for programming Bluetooth devices
- Reading data from Bluetooth *services* and *characteristics*
- Using the *specializations* for specific devices
- All the specializations for specific devices
- All the different BC BASIC programs that come with Best Calculator, IOT edition

## 34.2 INITIALIZING YOUR DEVICE AND AVAILABLE PROPERTIES

There are three stages of using a Bluetooth device. In the first stage (after calling `Bluetooth.Devices`), you can get all paired Bluetooth devices. This provides just a few simple properties and methods.

To get to stage two, call `device.Init()` on any particular device. This will verify that your program is allowed to access the device (the user can refuse permission), and that no other program has access to the device. Once you call `Init()`, you are allowed to read and write data to the device. You can call `Init()` as

often as you wish, but you must call it at least once to get full access to the device.

In the third stage, you call the *device*.As("device type") method to get a specialized version of the device. This gives you methods that are customized for a particular device and which are much easier to use.

Example of the second stage, showing the call to Init() and the additional methods available. This is a modification of the starting program; Init() is called before calling PrintBluetoothInfo and the PrintBluetoothInfo function is updated to display the device's Bluetooth address.

```
CLS BLUE
PRINT "Bluetooth Initialization"

REM
REM How many Bluetooth devices are available?
REM
devices = Bluetooth.Devices ()
PRINT "Count", devices.Count

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    device.Init()
    PrintBluetoothInfo (device)
NEXT i

FUNCTION PrintBluetoothInfo(bt)
    PRINT "NAME", bt.Name
    PRINT "ID", bt.Id
    PRINT "PROPERTIES", bt.Properties
    PRINT "ADDRESS", bt.BluetoothAddress
    PRINT "CONN.", bt.ConnectionStatus
    PRINT ""
RETURN
```

Once you initialize an object, you can read and write data. For example, if you have a DOTTI device from wittidesign.com, you can read the power level using *service* 180f and reading the Power data from *characteristic* 2a19.

### 34.2.1 Error handling and Bluetooth

You must handle Bluetooth errors if you want to create a robust program.

Bluetooth calls can fail in many ways: the computer that your program is run on might not have a Bluetooth radio; the devices might be out of range or turned off or might go out of range while your program is trying to communicate.

You should check the error value returned by the Bluetooth methods.

```
CLS BLUE
PRINT "Bluetooth Initialization"

devices = Bluetooth.Devices ()

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    status = device.Init()
    IF (status.IsError)
        PRINT "Unable to initialize", device.Name
    ELSE
        PrintBluetoothInfo (device)
    END IF
NEXT i

FUNCTION PrintBluetoothInfo(bt)
    PRINT "NAME", bt.Name
    PRINT "ID", bt.Id
    PRINT "PROPERTIES", bt.Properties
    PRINT "ADDRESS", bt.BluetoothAddress
    PRINT "CONN.", bt.ConnectionStatus
    PRINT ""
RETURN
```

### 34.3 THE OBJECTS YOU USE WHEN PROGRAMMING YOUR BLUETOOTH DEVICE

You will use four main objects when you program a Bluetooth device. If you've ever done object-oriented programming, you'll recognize the terms *method* and *property* as they are used in BC BASIC. BC BASIC has some simplifications that make BC BASIC a little different from what you've seen before.

A method is like a function call (like `SIN(0.01)`) but where the “function” is the name of a variable followed by a dot and followed by the method name to call. For example, to get a list of all of the available Bluetooth devices, you call the `Devices()` method on the globally available Bluetooth object like this: `LET devices = Bluetooth.Devices()`. In this example, a new variable `devices` is made by calling the `Devices()` method on the `Bluetooth` object.

A property is like a method but it doesn't take any arguments and you don't need any parenthesis. For example, once you have the list of Bluetooth devices you can find out its length with the `Count` property like this: `PRINT devices.Count.`

You will always start with a single global object (like `Bluetooth`) which is always available to your program. From there, you can get other objects like an array (list) of devices and the individual devices. Unlike other language, BC BASIC does not make you use the `new` operator just to make an object. BC BASIC also only lets you call a single method at a time.

For example, you must make these calls

```
devices = Bluetooth.Devices()  
device = devices.Get(1)
```

You cannot combine the two method calls, so that you cannot make a single line like `Bluetooth.Devices().Get(1)` and have it work.

BC BASIC does not allow you to define your own classes and objects in BC BASIC.

#### 34.3.1 The Bluetooth object

Subject to your license, the `Bluetooth` object is always available to your program. It contains just a few methods. The `PickDevicesName()` lets the user pick a single Bluetooth device. `Devices()` and `DevicesName()` methods both return a list of Bluetooth devices.

The **Bluetooth.Devices()** method returns a list of all paired Bluetooth devices on the system except for devices reserved to the system.

The **Bluetooth.DevicesName("name")** method returns the same list except that the Windows device name must match the passed-in value. The value must either exactly match or can start with or end with a star to match the end or start of a name. This is very useful when you're trying to control specific devices. For example, to control the DOTTI device, it's handy to use just the devices listed in **Bluetooth.DevicesName("\*Dotti")** because DOTTI devices by default are all called "Dotti" and when changed by the DOTTI app get a name that ends with -Dotti.

The name can also be a list of possible names to match, separated by a comma. This is useful when your device might be known by different names (e.g., the TI SensorTag 1350 can be known as the CC1350 SensorTag or the SensorTag v2.0)

The **Bluetooth.PickDevicesName("name")** method uses the same list that DevicesName returns and show a dialog that where the user can pick a single device.

### 34.3.2 The Bluetooth.Devices object (Array / ObjectValueList)

The list of available devices from Bluetooth.Devices and Bluetooth.DevicesName has a single property called Count and a single method called Get(<index>) which gets individual devices.

The *deviceList.Count* property returns the number of devices in the list. An empty list has length zero.

The *deviceList.Get(index)* method will return a single device. The index must be a value that is 1 or more and less than or equal to the Count property. For example, if you call Bluetooth.Devices and get a list whose count is 2, then you can call Get(1) and Get(2) to get the two devices.

If you call Get() with the wrong number of arguments or with an invalid argument (not a number, or out of range), the returned object is an error object.

### 34.3.3 Individual Bluetooth devices from Bluetooth.Devices

You will get individual Bluetooth devices by calling the Bluetooth.PickDevicesName() or by calling the Get() method on the list that you get by calling Bluetooth.DevicesName("name") or Bluetooth.Devices().

Devices start off uninitialized. From an uninitialized device, you can read the name and id (plus a little bit more). After you call Init() and the call succeeds, you can read and write data to the device using the Read and Write methods.

You can select SPP (Serial Port Protocol, or Rfcomm) devices with Bluetooth.PickDevicesNameRfcomm("<name>") or from Bluetooth.DevicesNameRfcomm("<name>") or Bluetooth.DevicesRfcomm().

Properties that are always available on a device include the Name, Id and Properties of the device. For developers who need very detailed information about their devices, you can also retrieve any property from the underlying Windows.Devices.Enumeration.DeviceInformation value that each device include. A list of these properties can be found on the MSDN web site.

If you've called the *device.Init()* method, then properties from the Microsoft Windows.Devices.Bluetooth.BluetoothLEDevice are also available. Where a property from the DeviceInformation conflicts with a property from BluetoothLEDevice, the DeviceInformation value is returned. When the property names are in conflict, you can access the BluetoothLEDevice property by prepending "BLE\_" to the property name. For example, *device.Name* is the DeviceInformation name, and *device.BLE\_Name* is the BluetoothLEDevice name.

The read and write routines are explained in the "Reading data from raw Bluetooth devices" section further down.

#### 34.3.4 Specializations

A number of common Bluetooth devices have *specializations* available using the *device.As("<type>")* method. The specializations let you control Bluetooth devices without having to know the exact services and characteristics and data formats for individual devices.

The specializations are each documented in a subsequent chapter. Specializations exist for different Bluetooth lights, gadgets and IOT sensor platforms like the TI SensorTag 2541.

### 34.4 RFCOMM (SERIAL-PORT) BLUETOOTH

Some Bluetooth devices (like the Infineon IFX\_NANOHUB and the Slant Robotics devices) work entirely using the Bluetooth Rfcomm mechanism. This is also called the Serial Port Profile (SPP).



The Rfcomm fundamental communication is that you can send data (often a string) to the device at any time, and it can send you information at any time. Often the data looks like it's individual packets – for example, you might send a command as one string and get a string in return. However, the underlying protocol is not one-for-one.

The little example sends a command to an Rfcomm device and prints the results.

```
bt = Bluetooth.PickDevicesRfcommName ("*")
bt.ReceiveString ("BtRecv")
bt.Send ("$hello\n")
FOREVER

FUNCTION BtRecv (device, data)
    PRINT "RECV:", data
RETURN
```

The \n in the string is converted to a Newline; a \r will be converted to a CR (Carriage-Return). Some Bluetooth devices require one and some require the other (and a few require both, generally as \r\n).

A note for people with advanced programming backgrounds: unlike some programming languages, the \n in a BC BASIC string is just a backslash and a 'n'; it's converted to a real carriage return or line feed right before sending to the device.

There are device.As ("name") specializations for selected Rfcomm devices like the IFX\_NANOHUB, Ardudroid and the Slant Robotics devices.

## 34.5 BLUETOOTH.WATCH (“SPECIALIZATION”, “FUNCTION”)

Bluetooth devices can be set to constantly broadcast “advertisements” which give a little bit of information about the status of the device. Google has a specification for what they call an “Eddystone” beacon which can broadcast short URLs which can point people to specific web sites.

Best Calculator, IOT edition, can listen for different types of Bluetooth broadcasts and call a function you specify when those broadcasts are detected.

Bluetooth.Watch(“specialization”, “function”) will set up a watch for Bluetooth advertisements that match the specialization and then calls the function callback. Specializations include “Bluetooth”, “Eddystone”, “Eddystone-URL”, and “Ruuvitag”.

### 34.5.1 Watch (“Bluetooth”, function)

The function will be called when any Bluetooth advertisement is detected. The function will be called with the Bluetooth address, the raw signal strength, and an array of DataSection data. The array will contain sub-arrays; each sub array will correspond to one data section. Each sub-array starts with the numeric code for the section type, and then one value per byte.

**Example callback:**

```
Bluetooth.Watch("Bluetooth", AllBluetooth)
REM more code here

FUNCTION AllBluetooth(address, rssi, data)
    Screen.ClearLines (3, 5)
    PRINT "Address", address
    PRINT "Signal", rssi
    PRINT "len", data.Count
RETURN
```

### 34.5.2 Watch (“Eddystone”, function)

The function will be called whenever a Bluetooth device advertises with an Eddystone type. Eddystone advertisements include data for service 0xFEAA.

The arguments to the function will be the Bluetooth address, the received signal strength indication (rssi) and the Eddystone frame type.

**Example callback:**

```
Bluetooth.Watch("Eddystone", Eddy)
```

REM more code here

```
FUNCTION Eddy(address, rssi, frameType)
Screen.ClearLines (5,7)
PRINT "Address", address
PRINT "Signal", rssi
PRINT "Type", frameType
RETURN
```

### 34.5.3 Watch("Eddystone-URL", function)

The function will be called whenever a Bluetooth device advertises an Eddystone-URL frame type. These frames include a short URL. For example, the RuuviTag generates short URLs where the fragment part is an encoded version of the current temperature pressure and humidity.

#### Example callback:

```
Bluetooth.Watch("Eddystone-URL", EddyUrl)
REM more code here
```

```
FUNCTION EddyUrl(address, ↴
    rssi, txpower, url)

Screen.ClearLines (5,9)
PRINT "Address", address
PRINT "Signal", rssi
PRINT "TX Power", txPower
PRINT "URL", url
RETURN
```

### 34.5.4 Watch ("RuuviTag", function)

The function will be called whenever a Bluetooth device advertises an Eddystone-URL frame type where the URL starts with <https://ruu.vi/#>. The fragment will be decoded into the advertised temperature pressure and humidity.

#### Example Callback:

```
Bluetooth.Watch("RuuviTag", ruuvi)
REM more code here
```

```
REM rss = Received Signal Strength Ind.  
FUNCTION ruuvi (address, rss, txpower, ↓  
    temperature, pressure, humidity)
```

```
Screen.ClearLines (3, 9)  
PRINT "Temperature", temperature  
PRINT "Pressure", pressure  
PRINT "Humidity", humidity
```

```
PRINT AT 7,1 "Address", address  
PRINT "RSSI", rss  
PRINT "TX", txpower
```

```
RETURN
```

## 34.6 SELECTING A DEVICE WITH PICKDEVICESNAMES AND MORE

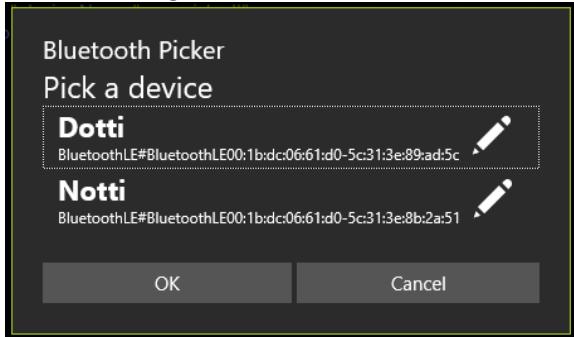
Often you need to be able to pick a specific device to control. For example, you might have several of the NOTTI illuminated devices. You want to pick which NOTTI device you want to control.

### 34.6.1 Bluetooth.PickDevicesName(<name pattern>)

Bluetooth.PickDevicesName(<name pattern>) is often a good choice. It will show a list of available Bluetooth devices that match your name pattern. For example, the NOTTI devices all have a name that ends with Notti. Your BASIC code would look like:

```
device = Bluetooth.PickDevicesName("*otti")  
IF (device.IsError)  
    PRINT "Sorry, no device was picked"  
ELSE  
    PRINT "Device ";device.Name;" was picked!"  
END IF
```

When PickDevicesName is called, it pops up a dialog to let the user pick a device. In the example, the user can pick any device that matches \*otti. This matches both the Notti and Dotti devices. If the user doesn't pick a device, taps cancel, or there is no matching device, the return value will be an error.



The dialog helps the user pick the correct device in two ways. Firstly, the device ID is listed; these are always unique to the actual device; they are never duplicated.

The user can set the “preferred” name of the device. The edit icon (.) lets the user set a ‘tag’ for a Bluetooth device. That tag is associated with the Bluetooth id; as long as the ID doesn’t change, the name will remain.

The name is roamed with Best Calculator, IOT Edition data (depending on your version of Windows). That means that when the user sets a name on one device, all their devices that are logged into the same MSA (Microsoft) account will have access to the same name. The name will be the same regardless of which BC BASIC program is running.

### 34.6.2 Bluetooth.DevicesName(<name pattern>)

Bluetooth.DevicesName(<name pattern>) lets you access the same list of devices that the PickDevicesName method shows the user. You can use this when you need to perform the same action on multiple devices.

### 34.6.3 Bluetooth.Devices()

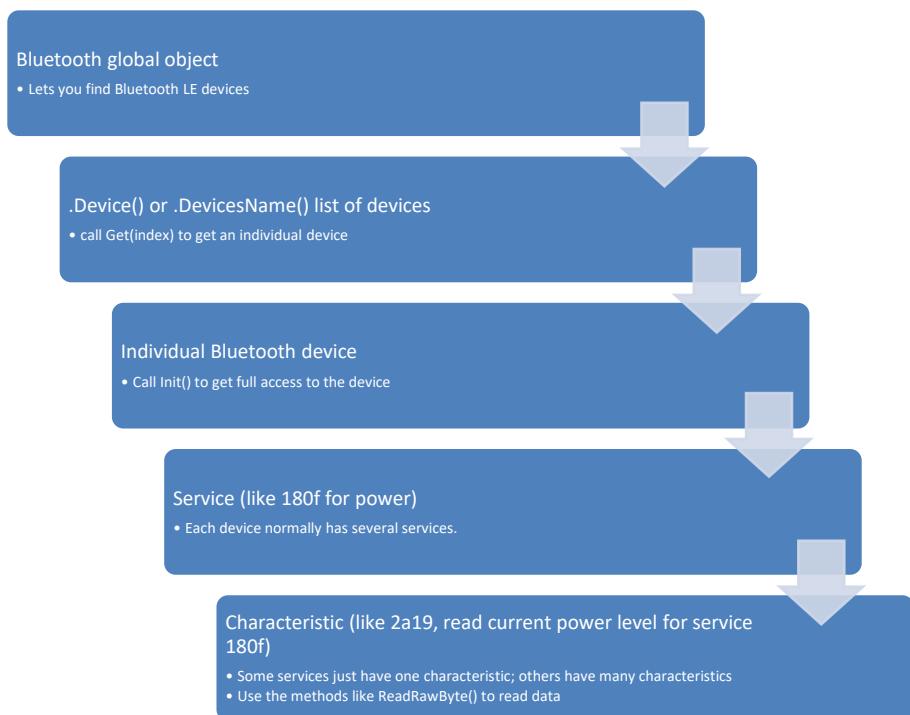
Bluetooth.Devices() returns a list of all the available Bluetooth devices regardless of their name.

Just because a device is returned doesn’t mean you have full access to the device. When you call the Init method for the first time, the user can choose to let BC BASIC have full access to the device, or can choose to not grant full access. Additionally, the device might already be used by another program.

### 34.7 READING DATA FROM RAW BLUETOOTH SERVICES AND CHARACTERISTICS

What you'll mostly be doing with a Bluetooth device is reading and writing to it. To do that, you will need to know the *service* and *characteristic* that you want to read or write. These take the form of long and short GUIDs. These will be documented in the device documentation somewhere. The data might be in a specific format that you will have to read. Additionally, when you read from a device you can read either the cached data (super fast, but not as fresh) or the raw data (always fresh, but much slower).

Every Bluetooth LE device exposes a set of *services*; each service is specified with a GUID. The Bluetooth functions all take in GUIDs as a string; the string is most commonly the short version of the GUID (e.g., 1800) but will also accept the long version (e.g., 00001800-0000-1000-8000-00805f9b34fb). Each service in turn exposes a set of *characteristics* which are also specified with a short or long GUID.



The hierarchy of Bluetooth features.

#### 34.7.1.1 *device.Read[Cached/Raw]Byte (service, characteristic) → byte*

There are two read methods that read a single byte from the device. Many characteristics have just a single byte of data, so it's handy to be able to just read that byte. The byte is read as an unsigned value from 0 to 255.

There are two separate Read methods available: ReadCachedByte and ReadRawByte. The ReadCachedByte method reads a cached byte; it's fast but the data may not be fresh. The ReadRawByte method asks the device for fresh data; it's likely to be slower but gets the most recent data.

#### 34.7.1.2 *device.Read[Cached/Raw]Bytes (service, characteristic) → array of data*

There are two read methods that read multiple bytes from the device. The data returned is an array (most likely a BCValueList).

The number of bytes read is available with the data.Count property

You can read individual bytes with the data.Get(index) method; the index is 1-based (1 up to and including Count). The data is returned as a unsigned byte with values 0 up to and including 255.

Example:

```
data = device.ReadRawBytes("2000", "2003")
PRINT AT 1,1 data.Get(1)
PRINT AT 2,1 data.Get(2)
PRINT AT 3,1 data.Get(3)
```

As a handy convenience, call data.GetValue(index, type) to interpret the data in a variety of ways. Supported types are:

- “int16-le” reads two bytes of data and treats them as a LSB and MSB of a two-byte, signed integer value. For example, if the two bytes are [4 10] the result will be  $(10 * 256) + (4)$ . If the second byte is more than 127, it's treated like a signed value. For example, if the two bytes are [255 255] then the returned value is -1.

There are two separate Read methods available: ReadCachedBytes and ReadRawBytes. The ReadCachedBytes method reads cached data; it's fast but

the data may not be fresh. The ReadRawBytes method asks the device for fresh data; it's likely to be slower but gets the most recent data.

Example: read the Power data from a DOTTI device using the raw Bluetooth read commands

```
CLS BLUE
PRINT "Read Bluetooth Power"

REM
REM How many Bluetooth devices are available?
REM
devices = Bluetooth.Devices()

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    IF (device.Name = "Dotti") THEN GetPowerInfo(device)
NEXT i

FUNCTION GetPowerInfo(bt)
    bt.Init()
    PRINT "NAME", bt.Name
    PRINT "POWER", bt.ReadRawByte("180f", "2a19")
    PRINT "CACHE", bt.ReadCachedByte("180f", "2a19")
    RETURN
```

Example: write a red dot to the DOTTI device in position (2,2) using the raw Bluetooth write commands

```
CLS BLUE
PRINT "Write red dot onto DOTTI device"

devices = Bluetooth.Devices()

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    IF (device.Name = "Dotti") THEN WriteDot(device, 10,
255, 0, 0)
NEXT i

FUNCTION WriteDot(bt, pos, r, g, b)
    bt.Init()
    REM The fff0 is the service for many DOTTI commands
    REM The fff3 is the characteristic used by service fff0
    REM for many of the DOTTI commands
```

```
REM the 7 and 2 are the bytes that define the DOTI  
REM command to send (0x0702 means set LED color)  
REM the pos is the position from 1 to 64  
REM the r g and b are the color to set.  
bt.WriteBytes ("ffff0", "fff3", 7, 2, pos, r, g, b)  
RETURN
```

## 34.8 USING CALLBACKS TO READ DATA

Instead of polling your device for data you can have the Bluetooth device tell you when the data changes. You have to perform two steps get callbacks:

Tell the device to send data with *device*.WriteCallbackDescriptor(service, characteristics, value)<sup>1</sup>. method. The service and characteristic say which data value to set the notifications on. The value parameter is one of 0 = None, 1=Notify and 2=Indicate. Your Bluetooth device specs will say whether any particular characteristic is Notify or Indicate capable. Many devices support Notify but not Indicate. The callback-name is the name of the BC BASIC function that you want to be called when the data changes; it's ignored when the value is 0.

You can place multiple callbacks on the same service and characteristic.

Example: getting notifications for accelerations on a TI SensorTag 2541

```
CLS BLUE  
PRINT AT 5,1 "Acceleration Data"  
  
devices = Bluetooth.DevicesName ("SensorTag*")  
  
REM  
REM Constants for TI SensorTag 2541 Accelerometer  
REM These are taken from the data sheets.  
REM  
AccService = "f000aa10-0451-4000-b000-000000000000"  
AccData = "f000aa11-0451-4000-b000-000000000000"  
AccConfig = "f000aa12-0451-4000-b000-000000000000"  
AccPeriod = "f000aa13-0451-4000-b000-000000000000"  
  
PRINT "COUNT", devices.Count
```

<sup>1</sup> This exactly corresponds with the Windows Runtime

WriteClientCharacteristicConfigurationDescriptorAsync method call. That name is much, much too long for a simple language like BC BASIC!

```
IF devices.Count < 1 THEN STOP
```

```
device = devices.Get(1)
```

```
PRINT "SensorTag Address", device.Init()
```

```
REM Tell the SensorTag to enable the Accelerometer
```

```
REM Config=1 means enable
```

```
REM Period=20 means get data fast (50 per second)
```

```
device.WriteBytes(AccService, AccConfig, 1)
```

```
device.WriteBytes(AccService, AccPeriod, 100)
```

```
REM 1=Notify (2=Indicate 0=None)
```

```
device.WriteCallbackDescriptor (AccService, AccData, 1)
```

```
device.AddCallback (AccService, AccData, "WriteAcc")
```

```
REM
```

```
REM Wait a little while and then turn off the Accelerometer
```

```
REM
```

```
FOR time = 1 TO 10
```

```
PAUSE 50
```

```
PRINT AT 1,1 time
```

```
NEXT time
```

```
REM
```

```
REM Turn off the accelerometer; turn off notify; remove  
callback
```

```
REM
```

```
device.WriteCallbackDescriptor (AccService, AccData, 0)
```

```
device.WriteBytes(AccService, AccConfig, 0)
```

```
device.RemoveCallback (AccService, AccData, "WriteAcc")
```

```
FUNCTION WriteAcc(device, x, y, z)
```

```
PRINT AT 3,1 " " " " "
```

```
PRINT AT 3,1 x, y, y
```

```
RETURN
```

### 34.9 USING THE SPECIALIZATIONS FOR SPECIFIC DEVICES

The easiest way to control a Bluetooth device is to use one of the specializations that are available for select Bluetooth devices. Specializations include methods that can easily control the devices without having to know service and characteristics GUIDs for your device. You would normally only use the Raw Bluetooth commands either when there isn't a specialization available for your device or when you need fine-grain control over your device.

To create a specialization, call the *device*.As("⟨device type⟩") method on a device object. Each available specialization is fully described along with the device type you need to provide.

It's important to know that BC BASIC doesn't verify that you are using the right specialization! That's because you might be controlling some new device, or a variant of an existing device. Using the wrong specialization will mostly just result in the commands not working.

Example: the (3,3) pixel to green using the DOTTI specialization.

```
CLS BLUE
PRINT "Write green dot onto DOTTI device"
devices = Bluetooth.DevicesName ("*Dotti")
FOR i = 1 TO devices.Count
    device = devices.Get(i)
    Dotti = device.As ("DOTTI")
    Status = Dotti.SetPixel (3, 3, 0, 255, 0)
    PRINT "status", Status
NEXT i
```

## 35 BLUETOOTH SPECIALIZATIONS FOR SPECIFIC DEVICES

---

Note that Best Calculator, IOT edition has no special relationship with any of the device manufacturers listed below. In all cases, the devices are programmed based on generally available information.

The Network Inspector program, also from Shipwreck Software, is useful when investigating any Bluetooth device. Most devices broadcast their capabilities in a way that any programmer can read and understand how to control these devices.

The appendix includes sample Bluetooth programs for many devices.

## 35.1 WORKING WITH ARDUINO

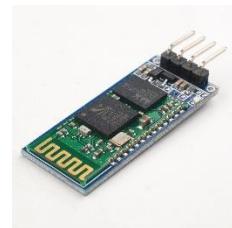
The Arduino is a very popular small microcontroller. Thanks to its simple but powerful IDE (integrated development environment), many libraries and low price, it's become perhaps the most common hobbyist microcontroller. You can find more information at <https://www.arduino.cc/>



Many Arduino devices are connected via a Bluetooth Rfcomm (also known as SPP, Serial Port Protocol) mini-board. Many are available from different vendors.

To communicate with an Arduino using a Bluetooth serial interface, follow the instructions in Rfcomm (Serial-port) Bluetooth. There are methods like

`PickDevicesRfcommName ("*")` to let your end-user pick a Bluetooth device to connect to and to send and receive data.



There isn't a single Bluetooth protocol for these connections. Many devices (like the Slant Robotics LittleBot and Infineon DPS310) have their own mini-protocol. Other boards use "standard" protocols like the Ardudroid.

If there's some protocol you would like supported, please contact [shipwrecksoftware@live.com](mailto:shipwrecksoftware@live.com) and let us know what protocol and what you're building!

## 35.2 ARDUDROID PROTOCOL

The Ardudroid protocol is a simple Rfcomm protocol for controlling an Arduino device. It includes commands for reading and writing the different pins. The Ardudroid specialization is built on an Rfcomm device.

The protocol documentation is at

<http://www.techbitar.com/ardudroid-simple-bluetooth-control-for-arduino-and-android.html>

. The Ardudroid specialization also includes two command extensions: ServoAttach and ServoMove to controlling servo motors. You need to download the ardudroid.ino file to your Arduino. It can also help to get the Ardudroid app from the Google Play store

Example: read a single Analog value

```
bt = Bluetooth.PickDevicesRfcommName("")  
ardu = bt.As ("Ardudroid")  
FOR i = 1 TO 10  
    val = ardu.Read (1, 1)  
    Screen.ClearLine (4)  
    PRINT "value", val  
    PAUSE 50  
NEXT i
```



In the example, we first pick an Ardudroid-compatible device to connect to. Then we read an analog value 10 times.

The available command are

Command	Meaning + Protocol
AnalogWrite (pin, value)	Write analog data to the pin *11 {pin} {value}
DigitalWrite(pin, value)	Write digital data to the pin *10 {pin} {value}
Read (pin, value)	Reads data from the Arduino. In the official version of the Ardudroid sketch, this always returns data from Analog pin 0 in the format Analog 0 = {value} *13 {pin} {value}
ServoAttach (servo, pin)	An addition to the protocol

	Attaches servo to pin using Servo.attach (pin) *14 {pin} {value}
ServoWrite (servo, value)	Writes servo data *15 {pin} {value}
Write (string)	Writes the exact string to the Ardudroid program. This is useful when you need to extend the Ardudroid protocol.

The original Arudroid sketch was updated to include Servo commands. An include statement, `#include <Servo.h>` was added to the top and definitions after `#define PIN_LOW 2`

```
// Additions to support Servo
Servo Servos[10];
#define CMD_SERVO_ATTACH 14
#define CMD_SERVO_WRITE 15
```

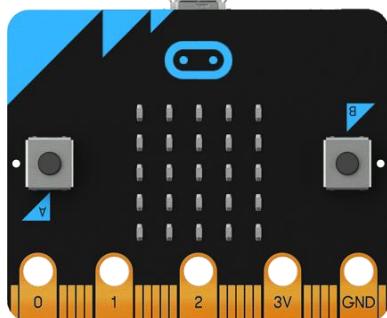
These additions were added to about line 103 near the end of the void loop() function.

```
if (ard_command == CMD_SERVO_ATTACH){
    Servos[pin_num].attach(pin_value);
    // the 'pin' is the servo number
}

// 'pin' is the servo number and 'value' is the value.
// often values are 0..180
if (ard_command == CMD_SERVO_WRITE) {
    Servos[pin_num].write (pin_value);
}
```

### 35.3 BBC MICRO:BIT

The BBC micro:bit is a small programmable computer designed with a set of on-board sensors plus easy connectivity to more devices through an expansion interface. For full information, please see the micro:bit web site at <http://microbit.org/> .



To pair the device, power the device on, press both the A and B button and while holding them down, press the reset button on the back. The device will show the string PAIRING MODE on the display. Then pair. A 6-digit code will be shown on the device.

Normally the device will run its out-of-box program and encourage people to press the buttons. Although the Bluetooth is functional in this mode, it's easier when the device is running a program that just does Bluetooth. A hex file called microbit-blue-pairing-not-required.hex that reprograms your BBC micro:bit to do just that is available at <http://www.bittysoftware.com/downloads.html> . A micro:bit uploader that will automatically move a hex file to your BBC micro:bit is available from [touchdevelop](#). The Bluetooth services are documented at <https://github.com/lancaster-university/microbit-docs/tree/master/docs/ble>

Your steps are:

1. Plug your micro:bit into your computer's USB port
2. Get a copy of the micro:bit uploader from touchdevelop and run it
3. Download a copy of microbit-pairing-not-required.hex from bittysoftware. Download it to your downloads directory; it will automatically up loaded to the micro:bit
4. The micro:bit will restart and ask you to draw a circle. This calibrates the magnetometer. Draw the circle by tipping the micro:bit around. Once you are done, the LED display will blank

Now you can pair the device from the Bluetooth Settings control panel.

The default Windows name for the device is BBC micro:bit; to get all of the devices call `devices = Bluetooth.DevicesName ("BBC micro:bit")`.

To get the BBC micro:bit specialization of a device, call tag = `device.As("microbit")` .

To list available methods, use tag.Methods

The specialization includes the following methods

Method	Description
GetName()	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from device.Name.
AccelerometerSetup(onoff, period, callback)  callback (tag, x, y, z)	Sets the accelerometer to on (1) or off (0); if on, then also sets the period. The callback is the name of the function to be called when the accelerometer data changes.  The period is in milliseconds. The x, y and z values are in "g" values.
ButtonSetup(onoff callback)  callback (device, A, B)	Sets the button callback to on (1) or off (0). The callback is the name of the function to be called when the button data changes.
MagnetometerSetup(onoff, period, callback)  callback (device, x, y, z)	Sets the magnetometer to on (1) or off (0); if on, then also sets the period. The callback is the name of the function to be called when the accelerometer data changes.  The period is in milliseconds.
TemperatureSetup(onoff, period, callback)  callback (device, temperature)	Sets the thermometer to on (1) or off (0); if on, then also sets the period. The callback is the name of the function to be called when the accelerometer data changes.  The period is in milliseconds.

	The temperature is in degrees Celsius.
SetLed (r1, r2, r3, r4, r5)	Sets the LED pattern on the micro:bit. The values are the 5 rows of LEDs. All-zeros will turn all the LEDs off; 31 (5 bits on) will turn all the LEDs on.
ToString()	Prints out a little information about your DOTTI device.
Write(string, speed)	Writes the string on the scrolling text. The speed is the speed in milliseconds; 100 is a good value.

The device is mostly programmed through special service ffb0. The SetColor call is characteristic ffb5, and takes in 4 bytes for red, green, blue and white values.

The device supports all the regular Bluetooth services and characteristics.

1800 Generic Access: 2a00 (Name) defaults to “BBC micro:bit”; 2a01 (Appearance) defaults to Unknown, 2a02 (Privacy) is False.

180a Device Info: 2a29 (Manufacturer, but the value is just “” instead of a specific value)

## 35.4 bELIGHT CC2540T LIGHT DEVELOPMENT KIT

### KIT

The beLight CC2540 device is a small high-intensity light development kit from Texas Instruments (TI). It has four built-in LEDs: red, green, blue and high-intensity white. For full information, please see the TI web site at <http://www.ti.com/tool/cc2540tdk-light>.



The Bluetooth PIN for pairing the device is 0.

The default Windows name for the device is beLight; to get all of the devices call `devices = Bluetooth.DevicesName ("beLight")`. To get the beLight specialization of a device, call `beLight = device.As ("beLight")`.

To list available methods, use `beLight.Methods`. The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>SetColor (r, g, b, white)</code>	Sets the color to a given red, green blue and white value.
<code>ToString()</code>	Prints out a little information about your device.

The device is mostly programmed through special service ffb0. The SetColor call is characteristic ffb5, and takes in 4 bytes for red, green, blue and white values.

The device supports all the regular Bluetooth services and characteristics.

1800 Generic Access: 2a00 (Name) defaults to “beLight”; 2a01 (Appearance) defaults to Unknown, 2a02 (Privacy) is False.

180a Device Info: 2a29 (Manufacturer, but the value is just “Manufacturer name” instead of a specific value)

## 35.5 DOTTI DEVICE

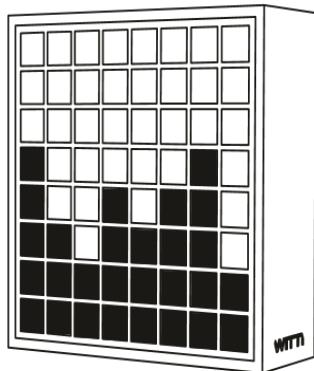
The DOTTI device is a desktop device with an 8x8 array of pixels. Each pixel can be programmed individually. For fully information, see the Witti Design web site at <http://www.wittidesign.com>.

The Bluetooth PIN for pairing the device is 123456.

The default Windows name for the device is Dotti; to get all DOTTI devices call `devices =`

`Bluetooth.DevicesName ("*Dotti")`. The

regular DOTTI app can rename the device but will always add a -Dotti to the end.



To get the DOTTI specialization of a device, call `Dotti = device.As("DOTTI")`. The name is in upper case to conform to how the manufacturer describes the device in their manual.

To list available methods, use `Dotti.Methods`

The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>GetPower()</code>	Gets the current battery power of the device using service 180f characteristic 2a19. The value is not cached.
<code>ChangeMode(mode)</code>	Sets the mode; 0=default on icon 1=animation 2=clock 3=dice game 4=battery indicator 5=screen off

LoadScreenFromMemory(part1, part2)	Loads the visible screen from memory. Part1 and Part2 describe the memory. There is a BASIC program that helps explain what these values should be.
SaveScreenToMemory(part1, part2)	Saves the current visible screen to memory.
SetAnimationSpeed(speed)	Sets the animation speed; 1 is very fast and 6 is very slow.
SetColumn (column, r, g, b)	Sets the given column to the given red, green and blue value. Columns are numbers 1 to 8.
SetName (name)	<p>Sets the Bluetooth name of the device as returned by service 1800 characteristic 2a00. The name will be modified as needed so that it ends with the word “-Dotti” (or is simply Dotti) to match the regular DOTTI app.</p> <p>The Windows name of the device may not change until the device is reset and the re-paired.</p>
SetNameArbitrary(name)	Like SetName, but the name won’t be changed to end with “-Dotti”
SetPanel (r, g, b)	Sets the entire panel color to the given red, green and blue values.
SetPixel (x, y, r, g, b)	Sets the given pixel to the given red, green and blue values. The x and y values must be 1 to 8.
SetRow (row, r, g, b)	Sets the given row to the given red, green and blue values. Rows are numbered 1 to 8.
SyncTime(h,m,s)	Sets the time on your DOTTI device.

ToString()	Prints out a little information about your DOTTI device.
------------	--

The special DOTTI service is fff0. Most commands are sent using characteristics fff3 except for and for the SET NAME command which uses characteristic fff5. By sending command bytes to these characteristics, you can control the DOTTI device.

The DOTTI device supports all the regular Bluetooth services and characteristics.

1800 Generic Access: 2a00 (Name) defaults to “Dotti”; 2a01 (Appearance) defaults to Unknown, 2a02 (Privacy) is False.

180a Device Info: 2a29 (Manufacturer, but the value is just “Manufacturer name” instead of a specific value)

180f Battery Level: 2a19 (Power, but it always seems to be 100)

fff0: D (fff3=D Data In): [writable], (fff5=C Command Channel)

## 35.6 HEXIWEAR WEARABLE PLATFORM

The Hexiwear from mikroElektronika (<http://hexiwear.com>) is a small wearable platform with weather, health and environmental sensors like accelerometers and pulse measurements. The device generates a unique pairing code each time it's paired.

As of September 2016, the device can pair with a Windows Phone but apparently does not pair with a Windows laptop or desktop.



The default Windows name for the device starts with HEXIWEAR; to get all of the devices call `devices = Bluetooth.DevicesName ("HEXIWEAR*")`.

To get the specialization of a device, call `tag = device.As("Hexiwear")`.

To list available methods, use `tag.Methods`

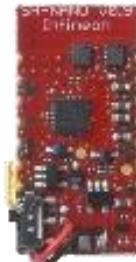
One of the unique things about the Hexiwear is that you can't control what data you can read. There are 4 modes (accessed via `device.ReadMode()`) . The 0=Idle 2=sensor tag 5=heart 6=pedometer. You can only read data when the user has manually set the device to the correct mode.

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached.
<code>GetManufacturerName()</code>	Gets the data from service 180a characteristic 2a29.
<code>GetFirmwareRevision()</code>	Gets the data from service 180a characteristic 2a29. The device I tested against reported being 1.0.1/1.0.0. This doesn't match the spec.
<code>GetPower()</code>	Reads a battery charge percent from 0..100 from service 180f characteristic 2a19.

GetMode()	Gets the Hexiwear mode. Sensors are only available in the right mode. 0 Idle 2 Sensor Tag 5 Heart 6 Pedometer (steps+calories)
GetAccelerometer()	Returns an XYZ value Example: LET d = tag.GetAccelerometer() PRINT d.X
GetGyroscope()	Returns an XYZ of the gyroscope values.
GetMagnetometer()	Returns an XYZ of the current compass setting.
GetLight()	Returns the current brightness value in a range of 0 to 100.
GetTemperature()	Returns the current temperature in degrees C.
GetHumidity()	Returns the current humidity as a percent from 0 to 100.
GetPressure()	Returns the current pressure in
GetHeart()	Gets the heart rate (when mode is 5) in beats per minute
GetSteps()	Gets the current steps count
GetCalories()	Gets the current calorie use estimate.

## 35.7 INFINEON DPS310

The Infineon DPS310 Pressure Sensor kit is a small, battery power combination pressure sensor, temperature sensor and altimeter mounted onto the IFX\_NANOHUB Bluetooth-enabled carrier board. <https://www.infineon.com/>



In the example, a raw Rfcomm (serial port) device is picked. The DPS310 is mounted on an IFX\_NANOHUB and shows up in Bluetooth with that name. Then the dps310 is created as a specialization of the raw Rfcomm device. Note that there's no good way for Best Calculator to know which device has which specialization for Rfcomm devices. You just have to write the code correctly.

```
CLS BLUE
PRINT "Altitude from the DPS310"
bt = Bluetooth.PickDevicesRfcommName ↓
    ("IFX_NANOHUB")
dps310 = bt.As ("DPS310", ↓
    "Dps310Altitude", "Dps310Pressure", ↓
    "Dps310Temp")

FOREVER

FUNCTION Dps310Altitude (dps310, value)
    Screen.ClearLine (4)
    PRINT "Altitude", value
RETURN

FUNCTION Dps310Pressure (dps310, value)
    Screen.ClearLine (5)
    PRINT "Pressure", value
RETURN

FUNCTION Dps310Temp(dps310, value)
    Screen.ClearLine (6)
    PRINT "Temp", value
RETURN
```

The sensor is supported with a generic Rfcomm Bluetooth interface, but it's much easier to use the DPS310 specialization. The specialization takes in up to three functions, one each for the altitude, pressure and temperature readings.

The altitude is in nominal meters above sea level, the pressure is in mBar and the temperature is in degrees Celsius.

When the specialization is made, three function names are passed in, one each for the altitude, pressure and temperature data. You can also just pass in a blank string ("") for any function that you don't need a callback for.

An explanation of the DPS310 raw Bluetooth is at

<http://sunriseprogrammer.blogspot.com/2017/07/infineon-sensor-hubfiguring-it-out.html> . An interesting set of contests for the Infineon was run by the

<http://Hackster.io> team; the set of samples is at

<https://www.hackster.io/contests/Infineon> .

## 35.8 MAGICLIGHT AND FLUX LIGHT

The MagicLight from Shultz and the Flux light are identical Bluetooth-enabled color-changing light bulbs.

The pairing code is 0.

The default Windows name for the device starts with LEDBlue; to get all of the devices call `devices = Bluetooth.DevicesName` ("LEDBLue\*").



**MAGIC LIGHT**  
THE WORLD'S SMARTEST LIGHT BULB

To get the specialization of a device, call `light = device.As("MagicLight")`.

To list available methods, use `light.Methods`

The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>SetColor (r, g, b)</code>	Sets the LED color to the given red, green and blue value. Columns are numbers 1 to 8.
<code>SetOff()</code>	Turns off the light
<code>SetOn()</code>	Turns on the light to the last color setting
<code>ToString()</code>	Prints out a little information about your device.

### 35.9 METAWEAR METAMOTION R DEVICE

The MetaMotion device is a very small wearable device in the mbientlab MetaWear range of sensors. There are several different MetaWear devices; with care this one device can be used with ones other than the MetaMotion.



The MetaMotion includes multiple sensors including ambient light, accelerometer, gyroscope and magnetometer and barometer. It's also got a built-in 3-color LED with a sophisticated programmable pulsing scheme. For full information, see the mbientlab.com web site at <https://mbientlab.com/> and a git repository at <https://github.com/mbientlab>.

No PIN is required for pairing.

The default Windows name for the device is MetaWear; to get all MetaWear devices call `devices = Bluetooth.DevicesName ("MetaWear")`

To get the MetaMotion specialization of a device, call `meta = device.As("MetaMotion")`. The name conforms to how the manufacturer describes the device in their manual.

To list available methods, use `device.Methods`. There are a complete set of examples for all these methods; just look for the BT: Metawear Metamotion sample.

The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>GetPower()</code>	Gets the current battery power of the device using service 180f characteristic 2a19. The value is not cached.

AccelerometerSetup (onoff, function[, gforce=2 [, rate=25]])	<p>When onoff is 1, will start to call function at the preferred rate.</p> <p>The accelerometer range will be set to the preferred value (or more); allowed values are 2, 4, 8 and 16 and are in units of G-force.</p> <p>The rate is in callbacks per second; the minimum value is .78125 and the maximum value is 3200.</p> <p>The callback function will be called with the Bluetooth specialization and the three values, x, y and z.</p>
AltimeterSetup (onoff, fnc, speed)	<p>When onoff is 1, set up the fnc called to be called when the altimeter data changes.</p> <p>The speed is in seconds between callback. Valid values are 4, 2, 1, 0.5, 0.25, 0.125 and 0.0625.</p> <p>The altimeter callback will be called with the Bluetooth specialization and the height in meters. Multiply the value by 3.2808399 to get the number of feet.</p> <p>You can have either altimeter or barometer data, but not both.</p>
BarometerSetup (onoff, fnc, [speed=1])	When onoff is 1, set up the fnc called to be called when the barometer data changes.

	<p>The speed is in seconds between callback. Valid values are 4, 2, 1, 0.5, 0.25, 0.125 and 0.0625</p> <p>The barometer callback will be called with the Bluetooth specialization and the atmospheric pressure in pascals.</p> <p>You can have either altimeter or barometer data, but not both.</p>
ButtonSetup (onoff, fnc)	<p>When onoff is 1, set up the fnc function callback then the button is pressed.</p> <p>The callback will be called with the Bluetooth specialization and the button value (1=pressed and 0=not pressed)</p>
GyroscopeSetup (onoff, function[, dps=500 [, rate=25]])	<p>When onoff is 1, will start to call function at the preferred rate.</p> <p>The dps (degrees per second) is the precision of the gyroscope. Allowed values are 125, 250, 500, 1000, and 2000</p> <p>The rate is in callbacks per second; the minimum value is 25 and the maximum is 3200</p> <p>The callback function will be called with the Bluetooth specialization and the three values, x, y and z.</p>
LedConfig (led, high, low, riseTime, highTime, fallTime, pulseLength, repeat)	Configure a single channel of the LED pattern. Led is 0=green 1=red 2=blue

	riseTime, highTime, fallTime and pulseLength are all in milliseconds repeat is the number of times to repeat the pattern.
LedOff()	Turns off the LED without deleting the pattern
LedOn()	Plays the LED pattern
SetColor (r, g, b)	Sets device color to the given red, green and blue value. This method will set the LED pattern and turn the LED on.
TemperatureRead()	Triggers a single temperature read.
TemperatureSetup(onoff,fnc)	When onoff is 1, sets up the fnc callback when the temperature is read. This will also trigger one temperate reading.  The function (fnc) will be called with the Bluetooth device and with a temperature reading in degrees Celsius.  Unlike the other sensors, you will not get a series of temperature callbacks. You must call TemperatureRead() to get a temperature value.
ToString()	Prints out a little information about your device.

The special MetaWear service is 326a9000-85cb-9195-d9dd-464cfbbae75a. Commands are sent using characteristic 326a9001-85cb-9195-d9dd-464cfbbae75a and data is read from characteristic 326a9006-85cb-9195-d9dd-464cfbbae75a

The device supports all the regular Bluetooth services and characteristics.

1800 Generic Access: 2a00 (Name) defaults to “MetaWear”; 2a01 (Appearance) defaults to Remote Control, 2a02 (Privacy) is False.

180a Device Info: 2a29 (Manufacturer. The value is MbientLab Inc

180f Battery Level: 2a19 (Power, but it always seems to be 100)

ffff0: D (ffff3=D Data In): [writable], (ffff5=C Command Channel)

Example program: make a single reading of the temperature data

```
device = Bluetooth.PickDevicesName ("MetaWear")
IF (device.IsError)
    CLS RED
    PRINT "No MetaWear device found"
    PRINT device
    STOP
END IF
meta = device.As ("MetaMotion")
IF (meta.IsError)
    CLS RED
    PRINT "Unable to connect to device"
    PRINT meta
END IF

CLS GREEN
PRINT "About my MetaWear device"
PRINT ""
PRINT "Name", meta.GetName()
PRINT "Man.", meta.GetManufacturerName()
PRINT "Power", meta.GetPower()
PRINT "Available Methods", meta.Methods
```

The program first finds the generic Bluetooth device and creates the specialization from the device. Then we can pull standard data from the device like the device name, manufacturer name and current power level.

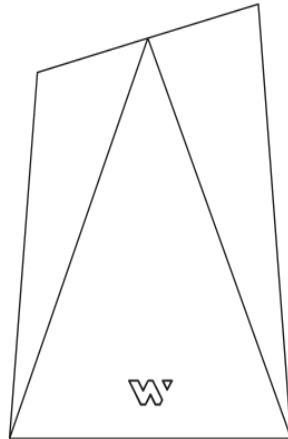
### 35.10 NOTTI DEVICE

The NOTTI device is a desktop device with a single light that can be set to any color. You can also program transitions and for colors changes to happen at a time in the future. For full information, see the Witti Design web site at <http://www.wittidesign.com>.

The Bluetooth PIN for pairing the device is 123456.

The default Windows name for the device is Notti; to get all NOTTI devices call **devices =**

**Bluetooth.DevicesName ("\*Notti")**



To get the NOTTI specialization of a device, call **Dotti**

**= device.As("NOTTI")**. The name is in upper case to conform to how the manufacturer describes the device in their manual.

To list available methods, use **Notti.Methods**

The specialization includes the following methods

Method	Description
<b>GetName()</b>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <b>device.Name</b> .
<b>GetPower()</b>	Gets the current battery power of the device using service 180f characteristic 2a19. The value is not cached.
<b>AlarmSetting (type, r, g, b, advance)</b>	Alarm settings. See <b>SetAlarmTime</b> for when the alarm will go off. Type is 0=off 1=every day 2=once only r, g, b is the color advance is how many minutes ahead of time to start changing.

	It's a value between 1 and 10; 1=2.5 minutes and 10=25 minutes.
ChangeMode(mode)	Sets the mode; 0=light on 1=light off 2=animation The mode doesn't seem to have any obvious effect.
SetAlarmTime(h,m)	Sets the time for the next alarm
SetColor (r, g, b)	Sets device color to the given red, green and blue value.
SetColorCustom(r1, g1, b1, r2, g2, b2)	Sets the device color to animate between color 1 (r1, g1, b1) and color 2 (r2, g2, b2)
SetName (name)	<p>Sets the Bluetooth name of the device as returned by service 1800 characteristic 2a00. The name will be modified as needed so that it ends with the word “-Notti” (or is simply Notti) to match the regular NOTTI app.</p> <p>Unlike the other NOTTI commands, SetName uses characteristic fff5, not fff3.</p> <p>The Windows name of the device may not change until the device is reset and the re-paired.</p>
SetNameArbitrary(name)	Like SetName, but the name won't be changed to end with “-Notti”
SyncTime(h,m,s)	Sets the time on your DOTTI device.
ToString()	Prints out a little information about your DOTTI device.

The special NOTTI service is ffff0. Most commands are sent using characteristics fff3 except for the SET NAME command which uses characteristic fff5. By sending command bytes to these characteristics, you can control the NOTTI device.

The NOTTI device supports all the regular Bluetooth services and characteristics.

1800 Generic Access: 2a00 (Name) defaults to “Notti”; 2a01 (Appearance) defaults to Unknown, 2a02 (Privacy) is False.

180a Device Info: 2a29 (Manufacturer, but the value is just “Manufacturer name” instead of a specific value)

180f Battery Level: 2a19 (Power, but it always seems to be 100)

ffff0: D (ffff3=D Data In): [writable], (ffff5=C Command Channel)

### 35.11 PUCK.Js

The puck.js device is a small JavaScript-powered platform from [puck-js.com](http://puck-js.com). The puck is designed by Gordon Williams, the designer of [Espruino](#), “JavaScript for Microcontrollers”.



The puck.js device is more flexible than most

Bluetooth sensor devices. This gives you more power and control, but also means you must work harder to set it up. You will have to write a JavaScript program on the puck.js device to send data to your BC BASIC program (which you also have to write).

The puck.js device emulates a UART with a TX and RX buffer (each of which is only 20 bytes long). To control the device, you send JavaScript commands to the device – for example, “LED1.set()\n” will turn the red LED on. You connect to the RX buffer with the puck.RxSetupLine(1, functionName) to get lines of data back, or the puck.RxSetup(1, functionName) to get individual strings from the puck as they arrive. The response can include an echo of your command, the JavaScript console prompt (“>”) and the results of the last statement (often “=undefined”). You can control this with the JavaScript echo(false) command.

There are two ways to download a JavaScript program to the puck.js device.

The best way is to use the Espruino Web IDE. You can also send the program to the puck.js using the puck.Tx(“string”) method (but watch out for those quotes!)

The puck.Js device includes a button, magnetometer, Infrared & RGB LEDs, thermometer and light sensors.

#### 35.11.1 Common Puck.Js JavaScript commands

To get data out of a puck.js device, you first program the device using JavaScript. In addition to the standard JavaScript statements and objects, the puck.js device includes additional objects and functions that are specific to the puck.js device.

Some of the commonly used additional objects and functions include:

JavaScript command	Notes
LED1.set()	Turn on red LED
LED1.reset()	Turn off red LED
LED2.set()	Turn on green LED
LED2.reset()	Turn off green LED

LED3.set()	Turn on blue LED
LED3.reset()	Turn off blue LED
BTN.read()	Returns either true or false.
Puck.mag()	Returns the current magnetometer value
Puck.light()	Return lux (light level) value
console.log(<string>)	Will print to the console (which shows up on the RX).
echo(false)	Turns off echoing of the input.
reset()	Resets the puck.js device; this helps keep the puck.js device from draining the battery.

### 35.11.2 Simplest puck.js program

Simplest possible puck.js program. The command “LED1.set()” command is sent to turn on the LED.

In this example

- First a raw Bluetooth device is selected by name. All puck.js devices show up with a name that starts with Puck.js
- Then the raw device is specialized as a puck object
- Then the Tx method on the puck object is called, passing in a simple JavaScript program that will turn on LED1 (the red LED).

```
CLS BLUE
PRINT "Turn puck.js LED1 on"

device = ↴
Bluetooth.PickDevicesName("Puck.js*")
IF (device.IsError)
PRINT "No device was picked"
ELSE
puck = device.As ("Puck.js")
status = puck.Tx ("LED1.set();\n")
REM The puck will reply to the command,
REM but this program won't pick it up.
END IF
```

### 35.11.3 A magnetometer program for the puck.js device

This magnetometer sample is more complex. The JavaScript program enables the magnetometer and, when the magnetometer data changes, will set the data to the BC BASIC program as a JSON object. The BC BASIC program listens for the JSON data, parses it, and displays it.

In addition, the program watches for button presses.

```
CLS BLUE
PRINT "Get Magnetometer data"

puckReset = "echo(false);\n"
puckProgram = "Puck.magOn(); Puck.on('mag',
function(xyz) { xyz.type='Mag';
console.log(JSON.stringify(xyz)); });\n"
puckButtonDownProgram = "setWatch(function() { var
value={'type':'Button', 'value':'Down'};
console.log(JSON.stringify(value));}, BTN, {edge:'rising',
debounce:50, repeat:true});\n"
puckButtonUpProgram = "setWatch(function() { var
value={'type':'Button', 'value':'Up'};
console.log(JSON.stringify(value));}, BTN, {edge:'falling',
debounce:50, repeat:true});\n"
puckOff = "echo(true);Puck.magOff();\n"

device = Bluetooth.PickDevicesName("Puck.js")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    puck = device.As ("Puck.js")
    puck.RxSetupLine (1, "Rx")
    REM The Puck works by sending JavaScript to the puck
    REM to be interpreted and run.
    Status = puck.Tx(puckReset)
    Status = puck.Tx(puckProgram)
    Status = puck.Tx(puckButtonDownProgram)
    Status = puck.Tx(puckButtonUpProgram)
    PRINT AT 3,1 "status", Status

    MAXTIME = 10
    FOR time = 1 TO MAXTIME
        PRINT AT 2,1 "TIME", time
        PAUSE 50
    NEXT time

    PRINT AT 3,1 "All done!"
    puck.Tx(puckOff)
    puck.RxSetupLine (0, "Rx")
```

```
END IF

FUNCTION Rx(puck, rx)
Screen.ClearLines (3, 4)
PRINT AT 3,1 "RX", rx
CONSOLE "RX:" + rx

REM data should be JSON
REM but stray output won't be
data = String.Parse("json", rx)
IF (data.type = "Button")
    PRINT AT 4,1 "Type", data.type
    Screen.ClearLine (6)
    PRINT AT 6,1 "Button", data.value
ELSE
    IF (data.type = "Mag")
        PRINT AT 4,1 "type", data.type
        Screen.ClearLines (7,9)
        PRINT AT 7,1 "X", data.x
        PRINT AT 8,1 "Y", data.y
        PRINT AT 9,1 "Z", data.z
    END IF
END IF
RETURN
```

## 35.12 RUUVITAG

The RuuviTag is a small, battery-powered sensor platform from [RuuviTag.com](https://ruuvi.com). The supported sensors include weather data (temperature, humidity and pressure)



The RuuviTag uses the Eddystone protocol to constantly broadcast the weather data; no pairing is needed.

Unlike the GATT based Bluetooth devices (like the TI SensorTag), the RuuviTag is connected using the `Bluetooth.Watch ("ruuvitag", function)` call. Your function is then called with temperature, pressure, humidity and radio power data.

### 35.12.1 Callback details

The callback takes these parameters

- Address      the Bluetooth address of the device
- RSSI strength      the received signal strength
- TX power      the transmitted signal strength
- Temperature      measured in degrees Celsius
- Pressure      measured in hPa
- Humidity      measured in percent

```
Bluetooth.Watch ("RuuviTag", "ruuvi")
MAXTIME = 10
FOR time = 1 TO MAXTIME
    Screen.ClearLine (1)
    now = DateTime.GetNow()
    PRINT now.Time
    PAUSE 50
NEXT time

FUNCTION ruuvi (address, rssi, txpower, ↴
    temperature, pressure, humidity)

    Screen.ClearLines (3, 9)
    PRINT "Temperature", temperature
    PRINT "Pressure", pressure
    PRINT "Humidity", humidity

    PRINT AT 7,1 "Address", address
    PRINT "RSSI", rssi
    PRINT "TX", txpower
RETURN
```

### 35.13 SKOOBOT

Skoobot is a tiny Bluetooth robot from  
<https://www.william-weiler-engineering.com/>.

Sample programs include the simplest possible program to set the robot into Rover mode, a keyboard-driven robot program, and a GUI program.



Use `device = Bluetooth.PickDeviceName("Skoobot")` to let the user select one particular Skoobot. Then to get a Skoobot specialization, use the `device.As` method, passing in "Skoobot" like this: `skoobot = device.As("Skoobot")`

When pairing, the Skoobot shows up as "Skoobot". No PIN code is needed.

Available Skoobot methods are

Category	Methods
Raw Motion Commands	<code>Left30()</code> <code>Right30()</code> <code>Forward()</code> <code>Backward()</code>
Stop	<code>Stop()</code>
Sounds	<code>PlayBuzzer()</code>
Data command	<code>SetupLight("function", ms)</code> <code>SetupDistance("function", ms)</code>
AI modes	<code>RoverMode()</code> <code>RoverModeRev()</code> <code>FotovoreMode()</code>

For example, to have the Skoobot move forward, once you have a skoobot specialization, just call `skoobot.Forward()`

Starting in version 3.20: the Skoobot specialization can provide a stream of distance and light values. You just have to pass in the function you want called and the number of milliseconds between callbacks.

The light callback gets a value in Lux. The distance callback gets a distance in centimeters (cm).

### 35.13.1 Skoobot Bluetooth commands

The Skoobot primary commands are using service 00001523-1212-efde-1523-785feabcd123 and characteristic 00001525-1212-efde-1523-785feabcd123. You might be familiar with these as they are the standard GUIDs for the Nordic Semiconductor nRF Blinky app. The meaning of the command byte has been extended so to include the Skoobot commands

Skoobot Command	Command Value
Right30	0x08
Left30	0x09
Right	0x10
Left	0x11
Forward	0x12
Backward	0x13
Stop	0x14
StopTurning	0x15
MotorsSleep	0x16
PlayBuzzer	0x17
RoverMode	0x40
FotovoreMode	0x41
RoverModeRev	0x42

The Skoobot specialization understands all these commands.

There are complete Skoobot sample programs including

**“A first control program”** which is the simplest possible real Skoobot program. It demonstrates how to connect to a Skoobot and to get a Skoobot specialization.

**“Control program for Skoobot”** is a GUI control program

**“Gopher-of-things for Skoobot”** shows how to control the Skoobot over the internet using the Gopher protocol to a local gateway. The local gateway runs the Gopher-of-things program, connecting to the nearby Skoobot using Bluetooth and offering up a Gopher interface to any Gopher client over the internet.

**“Keyboard-driven Skoobot program”** is a command-line program for the Skoobot.

**“Light and Distance”** demonstrates getting just light and distance data straight from the Skoobot and graphing the results. BC BASIC is particularly good at making it really simple to make graphs that automatically update themselves.

### 35.13.2 The simplest complete Skoobot program

```
CLS
PRINT "SKOOBOT CONTROL PROGRAM"
PRINT "Sets the Skoobot into Rover mode"
PRINT "Will automatically stop after 5 seconds"

device = Bluetooth.PickDevicesName ("Skoobot*")
IF (device.IsError) THEN
    PRINT "No device selected"
    STOP
END IF

REM get the specialization of the device
skoobot = device.As ("Skoobot")

skoobot.RoverMode()

REM Run for about 5 seconds
PAUSE 50*5
skoobot.Stop()

STOP
```

Important points:

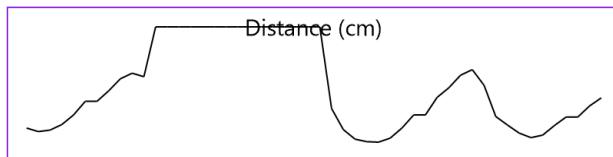
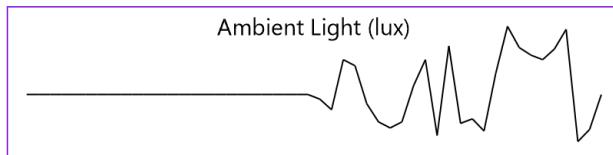
1. The **Bluetooth.PickDevicesName(“Skoobot”)** code lets the user select a Skoobot to control. The Skoobot\* string ensures that only Skoobot robots are shows in the selection list
2. The **IF device.IsError** code handles the case of no robot being found or the user canceling the selection
3. The **skoobot = device.As (“Skootbot”)** code get s *specialization* of the device. The device object just has pain Bluetooth commands available; the specialization has the robot commands.
4. The **skoobot.RoverMode()** code turns on Rover mode.
5. Lastly, **skoobot.Stop()** will stop the robot.

### 35.13.3 Example: complete program to graph Light data

This example is a complete program that demonstrates setting up an ambient light callback, adding data to an array, and automatically graphing that data.

SKOOBOT LIGHT AND DISTANCE PROGRAM

Ambient	64
Distance	14.478



```
CLS WHITE BLACK
PRINT "SKOOBOT LIGHT AND DISTANCE PROGRAM"
```

REM

REM Pick a Skoobot. If there's only one, select it  
REM automatically with no user intervention.

REM

devices = Bluetooth.DevicesName("Skoobot")

IF (devices.Count = 0)

    PRINT "No Skoobot devices found!"

END IF

IF (devices.Count = 1)

    device = devices[1]

ELSE

    device = Bluetooth.PickDevicesName ("Skoobot")

    IF (device.IsError) THEN

        PRINT "No device selected"

        STOP

    END IF

END IF

skoobot = device.As ("Skoobot")

REM Set up a Light array

DIM lightData()

lightData.MaxCount = 50

```
lightData.RemoveAlgorithm = "First"  
gl = Screen.Graphics (50, 50, 100, 400)  
gl.Title = "Ambient Light (lux)"  
gl.GraphY (lightData)
```

```
REM Set up a distance array  
DIM distanceData()  
distanceData.MaxCount = 50  
distanceData.RemoveAlgorithm = "First"  
gd = Screen.Graphics (50, 180, 100, 400)  
gd.Title = "Distance (cm)"  
gd.GraphY (distanceData)
```

```
REM  
REM Set up the Light and Distance functions.  
REM Each will be called back about every 100  
milliseceonds.
```

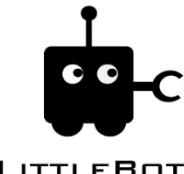
```
REM  
skoobot = device.As ("Skoobot")  
skoobot.SetupLight ("Light", 100)  
skoobot.SetupDistance ("Distance", 100)  
FOREVER WAIT
```

```
REM Called with light values in LUX  
FUNCTION Light (currskoobot, lux)  
    GLOBAL lightData  
    lightData.Add (lux)  
    Screen.ClearLine (2)  
    PRINT "Ambient", lux  
RETURN
```

```
REM Called with an approximate distance in cm  
(centimeter)  
FUNCTION Distance (currskoobot, cm)  
    GLOBAL distanceData  
    distanceData.Add (cm)  
    Screen.ClearLine (3)  
    PRINT "Distance", cm  
RETURN
```

### 35.14 SLANT ROBOTICS LITTLEBOT

The LittleBot from Slant Robotics is a small, cute, Arduino-based robot kit released in 2017 from a Kickstarter [campaign](#) and now supported on their [web site](#) at <http://littlearmrobot.com>. The brains of the robot are a small Arduino-based controller that can be controlled with a built-in Bluetooth serial port.



Before you can use the LittleBot you need to download the Walteros ([walteros\\_05.2.ino](#)) sketch for the robot from their [downloads](#) page. Install it using an Arduino IDE (Integrated Development Environment). When I programmed mine, I had to remove the Bluetooth board for the USB serial connection to work correctly. I also had to manually set the Arduino type (it's an "Arduino Nano")

The Bluetooth connection is "HC-06" and the PIN is **1234**

The robot's Bluetooth command language consists of a series of numeric commands with optional arguments. Each correct command is responded to with a single "d" character ("d" stands for "done").

The commands are:

Command	Meaning
1 <left><right><grip>	Set the robot wheel speed (the left and right) and the gripper position. Common speeds are 20 and 30 (the code says that 30 is "half speed")
222	Stops the motors. The gripper position doesn't change.
256	Autonomous mode. The robot will move around at will.

Unfortunately, there isn't a command that returns the current ultrasonic detection value.

A very simple program to control the robot. It connects to the robot, switches into autonomous mode for about 20 seconds, and then turns the robot off.

```
bt = Bluetooth.PickDevicesRfcommName ("*")
bt.ReceiveString ("BtRecv")
bt.Send ("256\n")
FOR time = 0 TO 10
    PAUSE 50
    now =DateTime.GetNow()
    Screen.ClearLine (2)
    PRINT "TIME", now.Time
NEXT time

FUNCTION BtRecv (device, data)
    CONSOLE "RECV:", data
RETURN
```

## 35.15 TI SENSORTAG 2541 (ORIGINAL VERSION)

The model 2541 SensorTag from Texas Instruments is a small, battery-powered sensor platform from TI. The sensors include an accelerometer, gyroscope, IR contactless thermometer, humidity sensor, magnetometer, barometer and on-chip temperature sensor.



The pairing code is 0.

The default Windows name for the device starts with SensorTag; to get all of the devices call `devices = Bluetooth.DevicesName ("SensorTag")`.

To get the specialization, call `tag = device.As("SensorTag2541")`.

To list available methods, use `tag.Methods`

The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>AccelerometerSetup(onoff, period, callback)</code>  <code>callback (tag, x, y, z)</code>	Sets the accelerometer to on (1) or off (0); if on, then also sets the period. The callback is the name of the function to be called when the accelerometer data changes. The period is in 1/100s of a second. The x, y and z values are in "g" values and range +/-2.
<code>BarometerSetup(onoff, period, callback)</code>	Sets the barometer to on (1) or off (0). If on, also sets the period. The callback is the

callback (tag, temp, pressure)	name of the function to be called with the data changes. The period is in 1/100s of a second; minimum value 10=100 ms. The temp is in degrees C. The pressure is in hectoPascal.
ButtonSetup(onoff, callback)  callback (tag, left, right, side)	Sets the Buttons to on or off. The callback is the name of the function to be called when the button data changes.
GyroscopeSetup (axis, period, callback)  callback (tag, x, y, z)	Sets the gyroscope to on (1 to 7) or off (0). The value says which axis to enable; 7 means x, y and z. If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second; minimum value 10=100 ms.
HumiditySetup (onoff, period, callback)  callback (tag, temp, humidity)	Sets the humidity sensor to on (1) or off (0). If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second; minimum value 10=100 ms. The temp is in degrees C. The humidity is relative humidity from 0 to 100.
IRSetup (onoff, period, callback)  callback (tag, object, ambient)	Sets the IR sensor to on (1) or off (0). If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second.

	Two temperatures are returned: the ambient (air) temperature and the contactless (object) temperature. Temperature is in degrees C.
MagnetometerSetup (axis, period, callback)  callback (tag, x, y, z)	Sets the magnetometer to on (1) or off (0). If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second; minimum value 10=100 ms.
ToString()	Prints out a little information about your device.

### Example program

```

devices = Bluetooth.DevicesNames ("SensorTag")
FOR i=1 TO devices.Count
    device = devices.Get(i)
    tag = device.As("SensorTag2541")
    tag.AccSetup(1, 100, "Acc")
NEXT i

FUNCTION Acc(tag, x, y, z)
    PRINT AT 1,1 x, y, z
RETURN

```

There are a number of sample programs in BC BASIC provided to demonstrate using the TI SensorTag.

## 35.16 TI SENSORTAG 1350 (2016 VERSION)

The model 1350 SensorTag from Texas Instruments is a small, battery-powered sensor platform from TI. The sensors include an accelerometer, gyroscope, IR contactless thermometer, humidity sensor, magnetometer, barometer and on-chip temperature sensor.



AS OF 2018, the SensorTag is no longer shipped with a TMP007 contactless IR thermometer. Although the chip is no longer placed on the SensorTag, it's still listed on the chip as a available sensor. There is no way to detect that the IR thermometer isn't present except that it doesn't produce any results.

No pairing code is needed. It will only pair with the most recent versions of Windows; it will not pair with the original Windows 10 or earlier operating systems.

The default Windows name for the device includes the word SensorTag; to get all of the devices call `devices = Bluetooth.DevicesName ("CC1350 SensorTag*", "SensorTag 2.0")`.

To get the specialization, call `tag = device.As("SensorTag1350")`.

To list available methods, use `tag.Methods`

The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>GetPower()</code>	Gets the current battery power of the device using service 180f characteristic 2a19. The value is not cached.
<code>AccelerometerSetup(onoff, period, callback)</code>	Sets the accelerometer to on or off (0); if on, then also sets the

callback (tag, ax, ay, az, mx, my, mz, rx, ry, rz)	period. The callback is the name of the function to be called when the accelerometer data changes. See below for the accelerometer on value.  The period is in 1/100s of a second.  The callback function returns the tag and three sets of X, Y, Z data. Accelerometer values are in "g" values. Magnetometer values are in micro-Tesla. Rotation (Gyroscope) values are in rotation degrees per second.
BarometerSetup(onoff, period, callback)  callback (tag, temp, pressure)	Sets the barometer to on (1) or off (0). If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second; minimum value 10=100 ms. The temp is in degrees C. The pressure is in hectoPascal.
ButtonSetup(onoff, callback)  callback (tag, left, right, side)	Sets the Buttons to on or off. The callback is the name of the function to be called when the button data changes.
HumiditySetup (onoff, period, callback)  callback (tag, temp, humidity)	Sets the humidity sensor to on (1) or off (0). If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second; minimum value 10=100 ms. The temp is in degrees C.

	The humidity is relative humidity from 0 to 100.
IO (value)	<p>Turns the device LEDs and buzzer on or off. The value is a bit-field, so each time you call this the on/off status of each device is updated.</p> <p>1=RED on 2=GREEN on 4=BUZZER on 0=all off</p> <p>The 1350 includes only a red LED; the 2650 has both a red and green LED.</p>
IRSetup (onoff, period, callback)  callback (tag, object, ambient)	<p>Sets the IR sensor to on (1) or off (0). If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second.</p> <p>Two temperatures are returned: the ambient (air) temperature and the contactless (object) temperature. Temperature is in degrees C.</p>
OpticalSetup (onoff, period, callback)  callback (tag, lux)	<p>Sets the light sensor to on (1) or off (0). If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second</p> <p>The callback returns the tag and the light level in lux.</p>
ToString()	Prints out a little information about your device.

The Accelerometer On value

The accelerometer On value is a complex. It's a bit field of the different sensors that you wish to enable. In general, the more sensors that are on, the more power the device takes.

1=Gyro Z axis	8=Accel. X axis	64=Magnetometer (all)
2=Gyro Y axis	16=Accel Y axis	(the Magnetometer does not allow you to turn on just one axis)
4=Gyro X axis	32=Accel Z axis	
7=Gyro all axis	56=Accel all axis	
128=enable Wake-on-motion	0=Accel range is 2G 256=Accel range is 4G 512=Accel range is 8G 768=Accel range is 16G	

For example, to turn on all of the sensors and set the accelerometer to a 2G range, add all of the gyro, accel axis and magnetometer values together:  
 $127 = 1+2+4+8+16+32+64$ .

#### Example program

```

device = Bluetooth.PickDevicesName ↓
("CC1350 SensorTag,SensorTag 2.0")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("SensorTag1350")
    PRINT AT 7,1 "SETUP", tag.ButtonSetup(1, "Button")

    FOR time = 1 TO 30
        PAUSE 50
        PRINT AT 3,1 "TIME", time
    NEXT time

    PRINT AT 8,1 "CLOSE", tag.ButtonSetup(0, "Button")
END IF

FUNCTION Button(tag, left, right, side)
    Screen.ClearLine(1)
    IF (left) THEN PRINT AT 1,1 "LEFT"
    IF (right) THEN PRINT AT 1,8 "RIGHT"
    IF (side) THEN PRINT AT 1,16 "SIDE"
RETURN

```

There are a number of sample programs in BC BASIC provided to demonstrate using the TI SensorTag. In addition to the simple samples, there's a fully-worked out Weather Station sample.

### 35.17 TI SENSORTAG 2650

The TI SensorTag is almost identical to the TI SensorTag 1350; see that device for details.

Caution: the original versions of the TI SensorTag 2650 did not pair with Windows. After updating the firmware using the [Debug DevPack](#), I could update the firmware to version 1.4, which does work with Windows.



AS OF 2018, the SensorTag is no longer shipped with a TMP007 contactless IR thermometer. Although the chip is no longer placed on the SensorTag, it's still listed on the chip as a available sensor. There is no way to detect that the IR thermometer isn't present except that it doesn't produce any results.

### 35.18 TI DISPLAY (WATCH)

The [TI Display \(Watch\) DevPack](#) is a Sharp [LS013B4DN04](#) LCD display connect to the TI SensorTag 2650 (1350).



The methods for the Watch DevPack to the TI SensorTag 1350 and TI 2650 specialization. Get the SensorTag specialization in the normal way and then use these additional methods. The methods are present even if the Display (Watch) is not attached.

**NOTE:** Setting up the device is finicky. You will have to update the firmware of the SensorTag, possibly using the [Debug DevPack](#) and force Windows to re-read the device characteristics, possibly using [Network Inspector](#) program by re-reading the device characteristics when the 'use cached values' checkbox is unchecked.

#### 35.18.1 Tag.WatchPrint(text)

Prints the text to the screen at the last cursor position by sending the string as bytes to characteristics ad01. The string is broken up in chunks of up to 16 bytes and each chunk is sent separately.

#### 35.18.2 Tag.WatchPrintAt(row, col, text)

Prints the text to the screen at the last cursor position by sending the string as bytes to characteristics ad01. The string is broken up in chunks of up to 16 bytes and each chunk is sent separately.

#### 35.18.3 Tag.WatchCls()

Clears the whole screen by sending command [3] and then move to [6, 0, 0] to characteristic ad02.

#### 35.18.4 Tag.WatchClearLine(line)

Clears a single line of the screen by sending command [4, <line>] to characteristic ad02.

#### 35.18.5 Tag.WatchInvert()

Inverts the screen (from black-on=silver to silver-on-black) by sending command [5] to characteristic ad02.

DEVPACK  
96x96  
that can  
(and the

are added  
SensorTag

**35.18.6 Tag.WatchOn()**

Turns the display on by sending command [2] to characteristic ad02.

**35.18.7 Tag.WatchOff()**

Turns the display off by sending command [1] to characteristic ad02. The display will then display the message, "Display off"

**Simple Clock example**

```
device = Bluetooth.PickDeviceName("SensorTag")
IF (device.IsError)
    CLS RED
    PRINT "No device was picked"
ELSE
    tag = device.As("SensorTag1350")
    tag.WatchCls()

    MAXTIME = 10
    FOR time = 1 TO MAXTIME
        now = DateTime.GetNow()
        tag.WatchPrintAt(4,3,now.Date)
        REM I only want hh:mm:ss (8 chars)
        tag.WatchPrintAt(6,4, LEFT(now.Time,8))
        PAUSE 50
    NEXT time
END IF
```

## 36 COMPLETE EXAMPLES

---

These example programs aren't just a dump of some code. They include explanations of how the different parts of the program work so that you can learn from them, understand them, and modify them to suite your needs.

### 36.1 CONNECTING TO MICROSOFT FLOW

Microsoft Flow is a cloud-based utility that performs actions (like sending email) based on triggers. This lets you automate your work flows. The cloud service is available at <http://flow.microsoft.com>

You can trigger your flows from Best Calculator using the `Http.Post` method on the `Http` specialization. There is a good blog post explaining how to trigger actions from an application at <https://flow.microsoft.com/en-us/blog/call-flow-restapi/>

Three key “gotcha’s” for Microsoft Flow: the flow trigger type is a **Request** flow and not the “http” triggers. You have to specify a **Content-Type**: `application/json` header when you POST your data; otherwise your data will be silently ignored. Lastly, to send email use the **Outlook.com service**, not the Office 365 Outlook service.



In this example temperature data from a MetaMotion device will be uploaded to Microsoft Flow and will cause an email to be sent. Only out-of-range data will be sent.

There are two big steps: you need to set up Microsoft Flow to accept your data (this is done through their web portal). And you need to make a little BC BASIC program that can trigger the flow.

#### 36.1.1 Set up the flow at Microsoft Flow

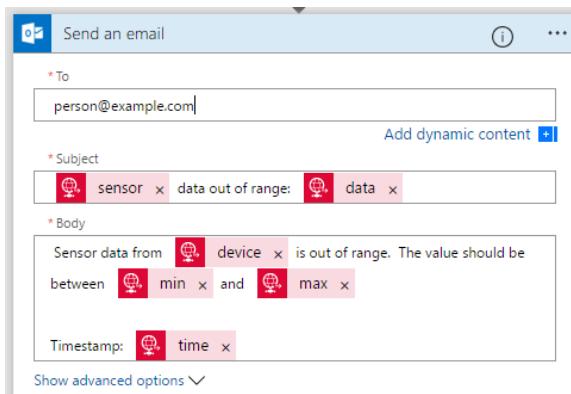
Your first step is to decide only your data format; you will need to provide your data schema when you create the flow. The data schema can be generated at

<http://jsonschema.net>. In this example I picked a simple data-logging schema; example data and the resulting schema look like this

Example JSON data	Resulting Schema
<pre>{   "data": 82.3,   "time": "2017-03-26 3:23:45.2",   "device": "outside #2",   "sensor": "temperature",   "min": 70,   "max": 80 }</pre>	<pre>{   "\$schema": "http://json-schema.org/draft-04/schema#",   "definitions": {},   "id": "http://example.com/example.json",   "properties": {     "data": {"id": "/properties/data", "type": "number"},     "device": {"id": "/properties/device", "type": "string"},     "max": {"id": "/properties/max", "type": "integer"},     "min": {"id": "/properties/min", "type": "integer"},     "sensor": {"id": "/properties/sensor", "type": "string"},     "time": {"id": "/properties/time", "type": "string"}   },   "type": "object" }</pre>

To create a flow at [flow.microsoft.com](https://flow.microsoft.com), and after you've signed up and are logged in:

1. Click “Create from blank” to make a new flow
  2. In the search box, enter “Request” and paste the schema into the request body
  3. Click “Add an action” to add the “sending an email” action. Select Outlook.Com as the service and Send an Email as the action.
- Surprisingly, I had to use the “Outlook.com” service, not the “Office 365 Outlook” service; I don’t know what the difference is, but one of them blocked me.



Now go back to the Request part of the flow. It will include a URL that you will be POSTing data to; this URL is automatically generated by the Flow service and will be monitored by them for data.

### 36.1.2 Write the BC BASIC program

The sample code has to:

1. Set up a URL to point to Microsoft Flow. Because these URLs are not intended for publication, the URL is stored in a named Memory cell; that way it's easily set and available to the program but it's not printed out in the listing.
2. Set up the monitoring value. The example is intended to show how you can monitor a value, so there are min and max values set up along with strings that describe the data.
3. Set up the sensor device. This example gathers data from the Mbient Labs MetaWear MetaMotion device, and in particular gathers temperature data. The MetaMotion device temperature sensor only sends data when it's poked (it doesn't provide a stream of updates)
4. Main loop to read the temperature data.
5. Temperature function is the callback function that will be called when the temperature data changes. You have to tell BC BASIC that this is a callback; it was done in part 3

```
meta.TemperatureSetup(1, "Temperature")
```

when the sensor device was set up.

6. SendData is called from the callback function when the readings are beyond the min and max values. SendData actually uploads data to the flow.microsoft.com site and is processed there.

SendData converts the raw data values into a correctly-formatted JSON value. This is made easy with String.Escape("json", list) method; that method takes an array of name/value pairs. You can fill in the name/value pairs with AddRow() method.

SetData also has to fill in a header value with a Content-Type:application/json header. Without this header, the JSON value will not be accepted by Microsoft Flow.

```
CLS BLUE
```

```
REM
```

```
REM The Microsoft Flow trigger URL is stored in the  
memory area
```

```
REM
```

```
memory = "Microsoft.Flow Example URL"
```

```
url = Memory.GetOrDefault (memory, "")
```

```
url = INPUT DEFAULT url PROMPT "Microsoft Flow URL"
```

```
Memory[memory] = url
```

```
REM
```

```
REM Set up the constant monitoring values
```

```
REM
```

```
min = 30
```

```
max = 40
```

```
deviceName = "My device"
```

```
sensor = "temperature"
```

```
REM
```

```
REM Set up the sensor device.
```

```
REM This program uses data from the MetaWear device
```

```
REM
```

```
device = Bluetooth.PickDevicesName ("MetaWear")
```

```
IF device.IsError
```

```
    CLS RED
```

```
    PRINT "No device picked"
```

```
    STOP
```

```
END IF
```

```
meta = device.As ("MetaMotion")
```

```
meta.TemperatureSetup(1, "Temperature")
```

```
REM
```

```
REM Main loop; will keep on spinning and
```

```
REM asking for updated temperature readings.
```

```
REM
```

```
ExitRequested = 0
```

```
MAXTIME=1000
```

```
FOR time=0 TO MAXTIME
```

```
    PAUSE 50
```

```
    meta.TemperatureRead()
```

```
    IF (ExitRequested > 0) THEN time = MAXTIME
```

```
NEXT time
```

```
REM
```

```
REM Callback when temperature changes
```

```
REM
```

```
FUNCTION Temperature(ble, celcius)
```

```
GLOBAL url
GLOBAL deviceName
GLOBAL sensor
GLOBAL min
GLOBAL max

time = DateTime.GetNow()
REM Convert to Fahrenheit
data = celcius * 9 / 5 + 32

Screen.ClearLine (9)
Screen.ClearLine (10)
Screen.ClearLine (11)
PRINT AT 9,2 "TIME", time.Time
PRINT AT 10,2 "TEMP", data

IF (data < min OR data > max)
    PRINT AT 11,1 "SENDING DATA"
    SendData (url, data, time, deviceName, sensor, min,
max)
    GLOBAL ExitRequested
    ExitRequested = 1
END IF
RETURN

REM
REM Format and send data to Microsoft Flow
REM
FUNCTION SendData(url, data, time, deviceName, sensor,
min, max)
REM
REM Put the data into correct JSON form
REM
DIM datalist()
datalist.AddRow ("data", data)
datalist.AddRow ("time", time)
datalist.AddRow ("device", deviceName)
datalist.AddRow ("sensor", sensor)
datalist.AddRow ("min", min)
datalist.AddRow ("max", max)
json = String.Escape ("json", datalist)

PRINT json

REM Microsoft Flow needs a header
REM Content-Type of application/json.
DIM header()
header[1] = "Content-Type: application/json"
result = Http.Post (url, json, header)
```

**RETURN result**

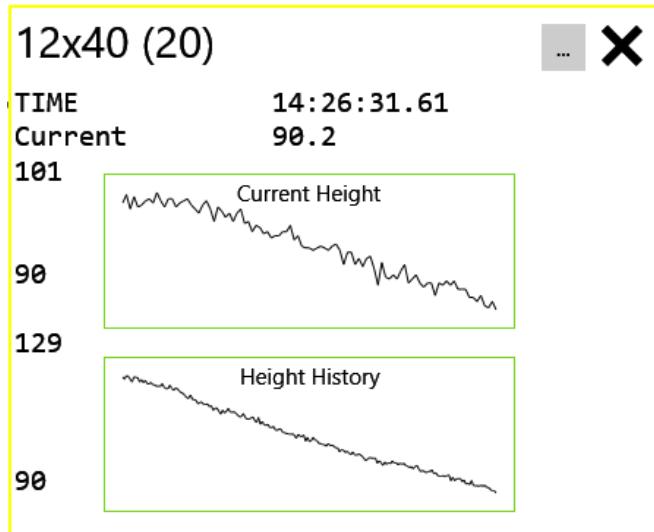
## 36.2 HIKING WITH AN ALTIMETER

A big part of the fun of supporting devices like the MetaWear is writing programs that do exactly what you want. When you hike with an altimeter, you can watch as you gain or lose altitude; it gives you a greater appreciation for the ups and downs of your trip.

These programs take the raw altimeter data from a MetaWear, convert it to feet, and adjust the height based on a “tare” (starting value), and then either just displays an instant result or graphs your entire trip.

### 36.2.1 The Graph program

The Graph program shows you two graphs: the top graph is a trace of your altitude for the last several minutes. The bottom graph is a summary of your altitude since your hike began.



Note: white and black were swapped on this graph for nicer printing.

The Current Height graph is “bumpy”: it contains the last 100 altitude measurements. The Height History graph is smoother because it’s covering a wider set of altitudes and because it’s a summarized history of the altitude.

This is a more complex version of the simpler Altitude program. Some points to note:

- This program uses the Automatic Graph feature of BC BASIC. This feature will automatically graph data from an array; you can update the array and the graph will be automatically updated. The currGraph and fullGraph are created and initialized from the currData and fullData arrays.
- The program also uses the MaxCount feature of the data arrays. We don't want the data arrays to grow infinitely. When you call data.Add() on an array with MaxCount set, the array will automatically remove data. For the currData array, the earlier data is removed (the data.RemoveAlgorithm is set to "First"). For the fullData array, a summary of all data is maintained using random *Reservoir Sampling* (the data.RemoveAlgorithm is set to "Random"). The fullData array is up to 200 elements long and has a reasonable summary of the entire data set.
- To make the graphs look a little nice, the min and max values of the array are printed on the screen. The screen is adjusted to fit onto a Lumia 650 phone.

```
REM
CLS BLUE
PRINT "Looking for a MetaWear.."
devices = Bluetooth.DevicesName ("MetaWear")
IF (devices.Count < 1)
    CLS RED
    PRINT "ERROR: no MetaWear devices found"
END IF
CLS BLACK

device = devices[1]
meta = device.As ("MetaMotion")

REM The program will stop when this is set to > 0
exitRequested = 0

REM Set up the curr and full data arrays and graph
DIM currData()
currData.MaxCount = 100
currData.RemoveAlgorithm = "First"

currGraph = Screen.Graphics()
currGraph.Title = "Current Height"
currGraph.SetPosition (60,60)
```

```
currGraph.SetSize(100, 275)  
currGraph.GraphY(currData)
```

```
DIM fullData()  
fullData.MaxCount = 200  
fullData.RemoveAlgorithm = "Random"
```

```
fullGraph = Screen.Graphics()  
fullGraph.Title = "Height History"  
fullGraph.SetPosition (60,185)  
fullGraph.SetSize(100, 275)  
fullGraph.GraphY(fullData)
```

```
REM  
REM Set up the altimeter  
REM  
meta.AltimeterSetup (1, "Altitude", 0.5)  
meta.ButtonSetup (1, "Button")
```

```
REM  
REM Main loop  
REM  
Screen.RequestActive()
```

```
10 REM LOOP_TOP  
  
IF (exitRequested > 0) THEN GOTO 20  
PAUSE 60  
dt = DateTime.GetNow()  
Screen.ClearLine (1)  
PRINT AT 1, 1 "TIME", dt.Time
```

```
GOTO 10  
20 REM LOOP_BOTTOM
```

```
REM  
REM All done; undo the setup  
REM  
Screen.RequestRelease()
```

```
msg="done!"  
meta.AltimeterSetup (0, "Altitude", 0.5)  
meta.ButtonSetup (0, "Button")
```

```
REM  
REM Altitude is called whenever altitude data comes in.  
REM  
FUNCTION Altitude(meta, height)
```

```
REM The main loop only exits about once per minute.  
When the  
    REM user presses the button to exit, they don't want to  
see the  
    REM graph keep on updating.  
    GLOBAL exitRequested  
    IF (exitRequested > 0) THEN RETURN  
  
    REM the meter-->feet conversion was copied from  
Bing.  
    currentRawAltitudeInFeet = height * 3.2808399  
    currentHeightInFeet = currentRawAltitudeInFeet -  
Memory.AltitudeTare  
  
    REM  
    REM Just Add'ing data to the arrays and doing a PAUSE  
    REM will upate the graphs on the screen.  
    REM  
    GLOBAL currData  
    GLOBAL fullData  
    currData.Add (currentHeightInFeet)  
    fullData.Add (currentHeightInFeet)  
    PAUSE 1  
  
    REM  
    REM Display some basis data on the screen.  
    REM  
    Screen.ClearLine (2)  
    Screen.ClearLine (3)  
    Screen.ClearLine (6)  
    Screen.ClearLine (8)  
    Screen.ClearLine (12)  
  
    PRINT AT 2,1 "Current", Math.Round  
(currentHeightInFeet, 1)  
    PRINT AT 3,1 Math.Round(currData.Max)  
    PRINT AT 6,1 Math.Round(currData.Min)  
    PRINT AT 8,1 Math.Round(fullData.Max)  
    PRINT AT 12,1 Math.Round(fullData.Min)  
  
RETURN  
  
FUNCTION Button(meta, value)  
    GLOBAL exitRequested  
    IF (value = 1) THEN exitRequested = 1  
RETURN
```

### 36.2.2 The Altitude program

The altitude program tells you, as quickly as possible, your current height. It assumes that you have only one Metawear sensor, so you don't need to pick it from a list. And it doesn't stay on; it gets a reading and then shuts down.

The entire altitude program has four sections. The first section picks an MetaWear device and sets it up to call a function called "Altitude" when the altitude data changes. The second section is the main loop; since the altitude data is sent to the Altitude function, we have to wait until some data shows up. The third section cleans up and tells the MetaWear to stop sending altitude data. This is important because otherwise it will waste its battery power sending data that you aren't listening to. The last section has the two functions Altimeter and Button. The Altimeter function converts the data to feet and display it. The Button function simply sets a global variable exitRequested; it's a quick way to exit the main loop.

```
REM
CLS GREEN
devices = Bluetooth.DevicesName ("MetaWear")
IF (devices.Count < 1)
    CLS RED
    PRINT "ERROR: no MetaWear devices found"
END IF

device = devices[1]
meta = device.As ("MetaMotion")

meta.AltimeterSetup (1, "Altitude", 0.5)
meta.ButtonSetup (1, "Button")

REM
REM The main loop. It will just go around a few
REM times and then exit.
REM
MAXTIME = 3
FOR time = 1 TO MAXTIME
    IF (exitRequested > 0) THEN time = MAXTIME
    PAUSE 60
    dt = DateTime.GetNow()
    Screen.ClearLine (1)
    PRINT AT 1, 1 "TIME", dt.Time
NEXT time

msg="done!"
```

```
meta.AltimeterSetup (0, "Altitude", 0.5)
meta.ButtonSetup (0, "Button")
```

```
REM called when new Altimeter data comes in.
FUNCTION Altitude(meta, height)
    currentRawAltitudeInFeet = height * 3.2808399
    currentHeightInFeet = currentRawAltitudeInFeet -
Memory.AltitudeTare
    PRINT AT 3,1 "Current", Math.Round
(currentHeightInFeet, 1)
RETURN

FUNCTION Button(meta, value)
    GLOBAL exitRequested
    IF (value = 1) THEN exitRequested = 1
RETURN
```

### 36.2.3 The TARE program

The Tare program reads the current altimeter data and asks you for your actual height. The difference is put into a named memory cell (“AltimeterTare”). This initializes your altimeter zero point; the other programs will use the ‘tare’ value to adjust the raw altimeter data into your actual height.

From Wiktionary: the Tare includes this definition:

*Verb (sciences) To set a zero value on an instrument (usually a balance) that discounts the starting point.*

I’m using tare in the scientific sense: it’s the zero value of the altimeter.

The air pressure constantly changes during the day. You’ll see that at the end of the hike, when you’re back at your original location, that the height is now “off”. That’s because the air pressure has changed during your hike. The height reported by the altimeter can be very different from your actual height.

```
REM
devices = Bluetooth.DevicesName ("MetaWear")
IF (devices.Count < 1)
    CLS RED
    PRINT "ERROR: no MetaWear devices found"
END IF
```

```
device = devices[1]
meta = device.As ("MetaMotion")

currentHeightInFeet = INPUT DEFAULT 203 PROMPT
"What is your elevation in feet?"
exitRequested = 0
meta.AltimeterSetup (1, "Tare", 0.5)

REM Wait for a callback
MAXTIME = 100
FOR time = 1 TO MAXTIME
  IF (exitRequested) THEN time = MAXTIME
  PAUSE 60
  PRINT AT 2, 1 "TIME", time
NEXT time

IF (time = MAXTIME)
  PAPER RED
  PRINT "Sorry, could not get the elevation"
ELSE
  PAPER GREEN
  PRINT "Adjust", Memory.AltitudeTare
END IF

result = Memory.AltitudeTare

FUNCTION Tare(meta, height)
  GLOBAL currentHeightInFeet
  REM Pasted from BING
  currentRawAltitudeInFeet = height * 3.2808399
  Memory.AltitudeTare = currentRawAltitudeInFeet -
  currentHeightInFeet
  PRINT AT 3,1 "Got an altitude"
  PRINT AT 4,1 "Current", currentHeightInFeet
  PRINT AT 5,1 "Raw", currentRawAltitudeInFeet
  PRINT AT 6,1 "Adjust", Memory.AltitudeTare
  GLOBAL exitRequested
  exitRequested = 1
RETURN
```

### 36.3 THE HAPPY BIRTHDAY PROGRAM

Best Calculator BASIC is able to make fun little animated programs. The Happy Birthday program in the EX: PLAY <music> package shows how to draw a little birthday cake and then, while the music is playing, makes the flames on different candles “flicker” in time to the music.

Here's the cake (chocolate, my favorite)



And here is the program

Some useful notes: the flames array (DIM flames()) holds the yellow ellipses that are the candle flames. The “normal” height is drawn to start with (the flames array is filled with the smaller flames in the DrawCake() function). Then the y position of a normal flame is captured in the normaly1 and normaly2 values and used to create the jumpy1 and jumpy2 values. The “jump” y1 and y2 are the value when that flame is selected.

I happen to know that the Happy Birthday song has exactly 9 different notes, which is why there are 9 different candles. I also happen to know what MIDI note each one is. I know the MIDI note because I printed them all out on the console and then examined them by hand.

The PLAY ONNOTE “onPlay” command means that each time a note is played, the onPlay function is called with information about the note. When it's called, it will set the flames to all be normal, wait a little bit, and then make one of the flames “jump”.

```
CLS BLUE  
PRINT "HAPPY BIRTHDAY"
```

```
g = Screen.Graphics (50, 50, 200, 400)
ncandle = 9
DIM flames()
DrawCake (g, n candle, flames)

flame = flames[1]
normaly1 = flame.Y1
normaly2 = flame.Y2
jumpy1 = normaly1 + 10
jumpy2 = normaly2+30

REM Call the onPlay function for each note.
PLAY ONNOTE "onPlay"
PLAY "T240 I20 L4 C# C# D C# F# F2 C#C# D# C# Ab F#2 C#
C# >C# <Bb F# F D#2 B Bb F# Ab F#2"
PLAY WAIT
AllFlamesNormal()

FUNCTION AllFlamesNormal()
    GLOBAL flames
    GLOBAL normaly1
    GLOBAL normaly2
    FOR f = 1 TO flames.Count
        flame = flames(f)
        flame.Y1 = normaly1
        flame.Y2 = normaly2
    NEXT f
RETURN

FUNCTION FlameJump(f)
    GLOBAL flames
    GLOBAL jumpy1
    GLOBAL jumpy2
    flame = flames(f)
    flame.Y1 = jumpy1
    flame.Y2 = jumpy2
RETURN

FUNCTION DrawCake (g, n candle, flames)
    REM Set a nice background
    g.Background = "#AAAAAA"

    w = g.W / 2
    h = g.H / 4

    g.Fill = "#4e2e28"
    g.Stroke = g.Fill
    x1 = (g.W - w) / 2
```

```
x2 = g.W - x1  
cx = (x1 + x2) / 2  
y1 = 40  
y2 = y1 + h
```

```
REM Make the cake  
g.Rectangle (x1, y1, x2, y2)  
g.Circle (cx, y1, w/2, h/2)  
g.Stroke = BLACK  
g.Circle (cx, y2, w/2, h/2)
```

```
REM Add the candles  
gapw = w / (ncandle+1)  
candlew = gapw * .66  
candleh = h * .95  
candlew = Math.Min (candlew, candleh / 4)  
g.Fill = BLUE
```

```
flamer = 10
```

```
FOR i = 1 TO ncandle  
    x = x1 + (i*gapw)  
    cx1 = x - candlew/2  
    cx2 = cx1 + candlew  
    cy1 = y2  
    cy2 = cy1 + candleh  
  
    g.Fill = "#EEEEEE"  
    g.Rectangle (cx1, cy1, cx2, cy2)  
  
    g.Fill = "#e1ad21"  
    flame = g.Circle (x, cy2+flamer+3, candlew/2,  
    flamer)  
    flames.Add (flame)  
NEXT i  
RETURN
```

```
REM I happen to know exactly what notes are  
REM in "Happy Birthday"
```

```
FUNCTION NoteToCandle(note)  
    IF (note = 61) THEN RETURN 1  
    IF (note = 62) THEN RETURN 2  
    IF (note = 63) THEN RETURN 3  
    IF (note = 65) THEN RETURN 4  
    IF (note = 66) THEN RETURN 5  
    IF (note = 68) THEN RETURN 6  
    IF (note = 70) THEN RETURN 7  
    IF (note = 71) THEN RETURN 8  
    IF (note = 73) THEN RETURN 9
```

```
RETURN 5  
RETURN
```

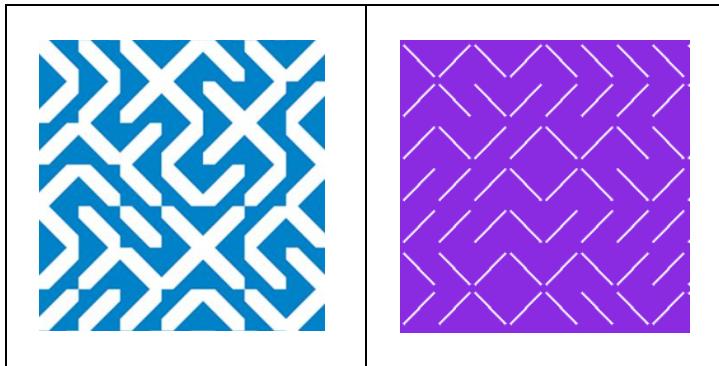
```
FUNCTION onPlay (note, instrument, duration, str)  
    candle = NoteToCandle (note)  
    Screen.ClearLine(2)  
    REM PRINT note, candle, instrument, duration  
  
    AllFlamesNormal()  
    PAUSE 10  
    FlameJump (candle)  
RETURN
```

To draw the cake, I draw a rectangle for the body of the cake and two ellipses (flattened circles) to make it look round. The color "#4e2e28" was picked by doing a Bing search for “html color chocolate”.

The candles are rectangles. If there are 9 candles that means that there must be 10 spaces total, so I just divide the cake width by 10; that gives me the center of each candle.

### 36.4 10 PRINT CHR\$(205.5+RND(1)) ;: GOTO 10

There's a famous program from the micro computer era. It prints a series of block character on the screen that form a sort of maze. There's even a web site, <http://10print.org> and book that dives into the little program and it's impact on the Commodore 64 computer.



*Commodore version (left) and BC BASIC version (right)*

BC BASIC doesn't allow for one-line programs and uses a different character set. A similar program for BC BASIC is

```
10 PRINT CHR$(0xFF0F+Math.Round(RND)*0x2D);  
20 GOTO 10
```

The chief difference is the choice of character sets. The Commodore uses "PETSCII" where characters 205 and 206 are the forward and reverse slashes. They look good on the Commodore's chunky display. The BC BASIC version uses the Unicode Fullwidth Solidus and Fullwidth Reverse Solidus. These don't have the nice property of being right next to each other; they are separated by hex 2D characters (45 decimal)

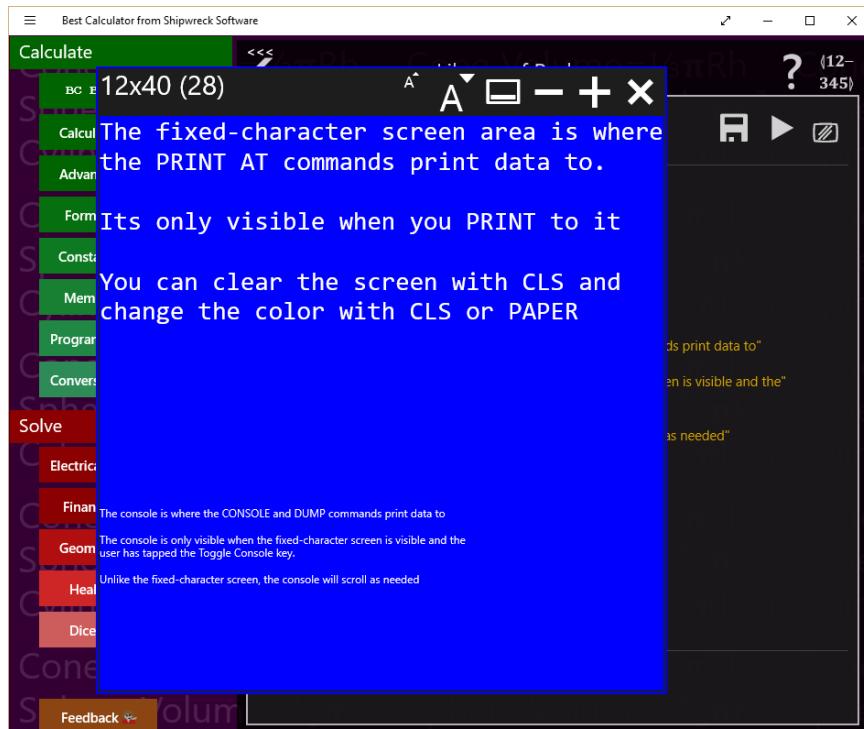
## 37 TEXT AND BC BASIC

BC BASIC has two ways to print to the screen. The main way is the PRINT and CLS and PAPER commands; they print to the “Fixed-character” screen. The CONSOLE and DUMP commands write to the console.

In addition, there are graphics available through the Screen.Graphics extension.

In the example, the area of the screen in blue with large type is the fixed-character screen. It's called that because each character prints at the same width. This lets you make tables and diagrams more easily.

Underneath is the console. The primary use of the console is debugging.



Normally the console is not visible.

## 37.1 FIXED CHARACTER SCREEN COMMANDS

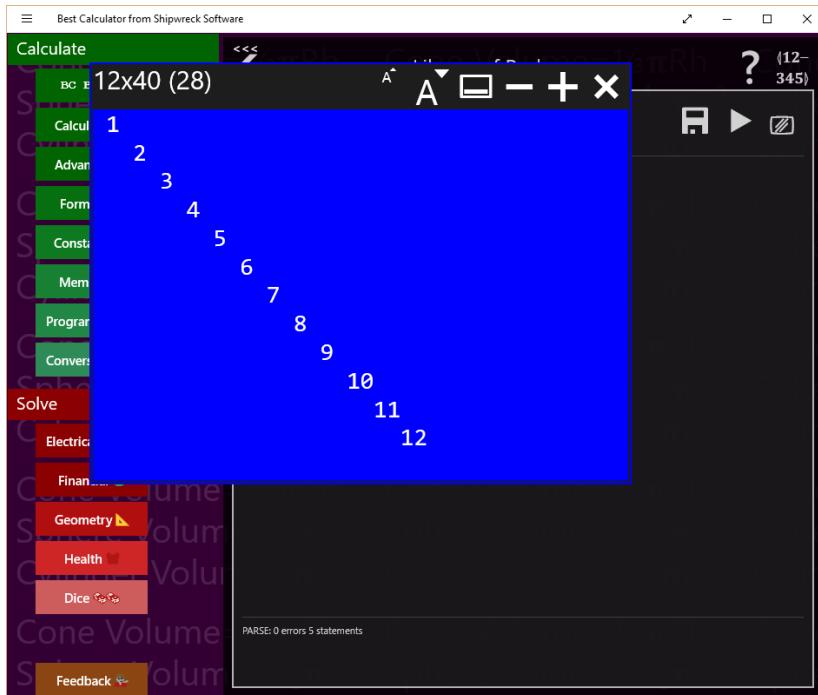
PRINT [AT line, column] <expression>

A comma between expressions will print each expression at 16-character positions

**Simple example:**

```
CLS BLUE
R = 1
10 PRINT AT R, 2*R R
R = R + 1
IF (R < 20) THEN GOTO 10
```

The following output is produced:



**Example that prints circles on the screen:**

```
REM
REM PLOT 3 CIRCLES
REM

CHAR = 1
R = 8
CX = 15
CY = 12
GOSUB 1000

CHAR = 2
R=5
GOSUB 1000

CHAR=3
R=13
CX=30
CY=15
GOSUB 1000

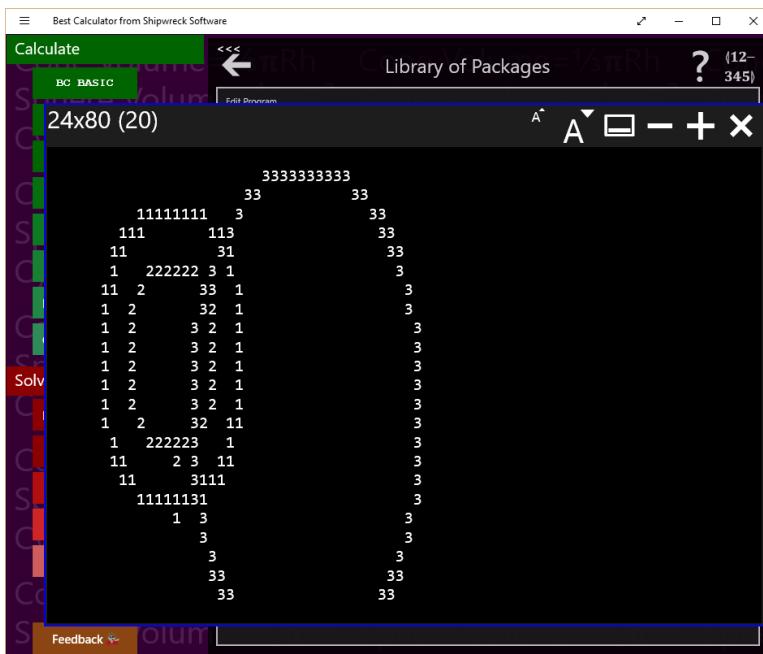
STOP

1000 REM DO A CIRCLE USING CHAR R CX AND CY
S = 0
1010 REM TOP OF LOOP
COL = R * SIN(S) + CX
ROW = R * COS(S) + CY
PRINT AT ROW,COL CHAR
S = S + 0.05
IF (S < 7) THEN GOTO 1010
RETURN
```

## Guide to Using Best Calculator

The circle program makes this output

Page | 290



## 37.2 CONSOLE COMMANDS

The console commands work on the console, a scrolling list of small-font output. The primary console commands are CONSOLE (writes to the console), DUMP (writes the name and value of all of the BC BASIC variables to the console) and the CLS and PAPER commands (which clear and possibly change the color).

**Example of using CLS to clear the screen and change the color:**

```
CLS YELLOW
```

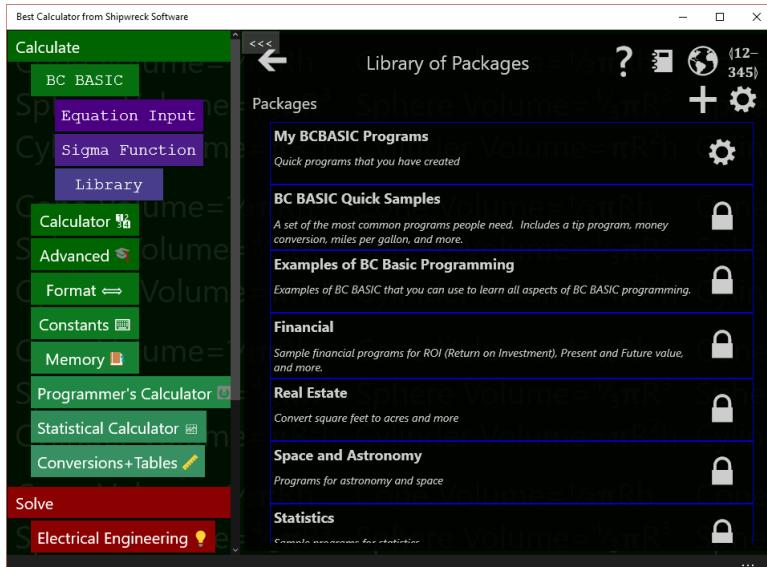
## 38 USING THE LIBRARY, STEP BY STEP

Once you have written a simple program, you might want to write more programs, and keep them all. BC BASIC includes simple Library functionality to keep all your programs. The Library also includes a series of sample programs for you to use.

In this first example, we'll write a simple program to convert square feet to acres. The steps are listed in the diagram and will be described in detail in each section.



### 38.1 ADD A NEW PACKAGE FOR YOUR PROGRAM



In this example, you will create a new program. First you need to select a package to put your program in.

Press the BC BASIC key and select Library. The *Library of Packages* screen will pop up. A package is a bundle of individual *programs*; you're going to make a single new *package* that contains a simple *program*. The program will convert from square feet to acres.

First, you need to make the *package* that your programs will be part of.

Tap the + key to add a new package. A new package will be created with a default name of "New Package".

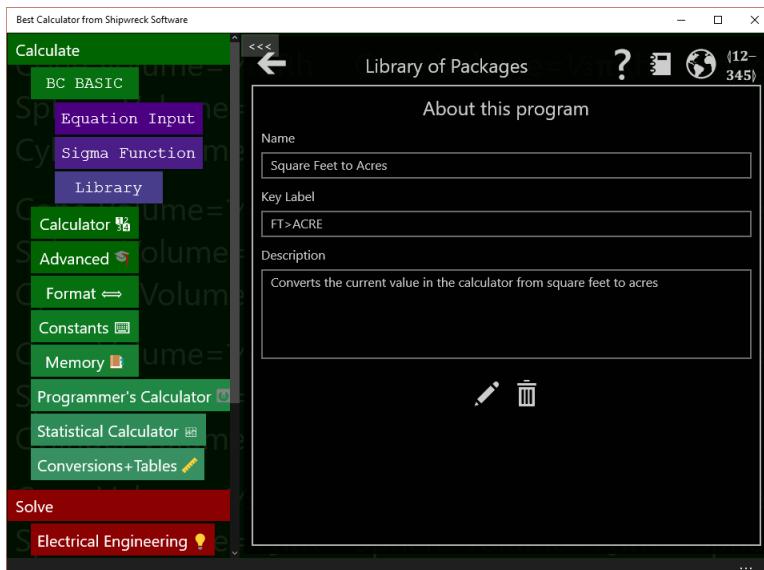
After you've written more programs and want to create more packages, you'll probably want to rename this package. Do that by tapping the

package's GEAR key (  ). Then change the name and description. The changes take place right away. Tap the BACK ARROW to get back to the list of packages.

## 38.2 ADD A NEW PROGRAM TO YOUR PACKAGE

Your new package is now ready for you to add your new program. Tap the package to see the list of programs in the package, and then tap the + to add a new program. It will be given the name NewProgram . It's also got a description and some code.

Tap the program's GEAR (  ) key to bring up the About this program screen.



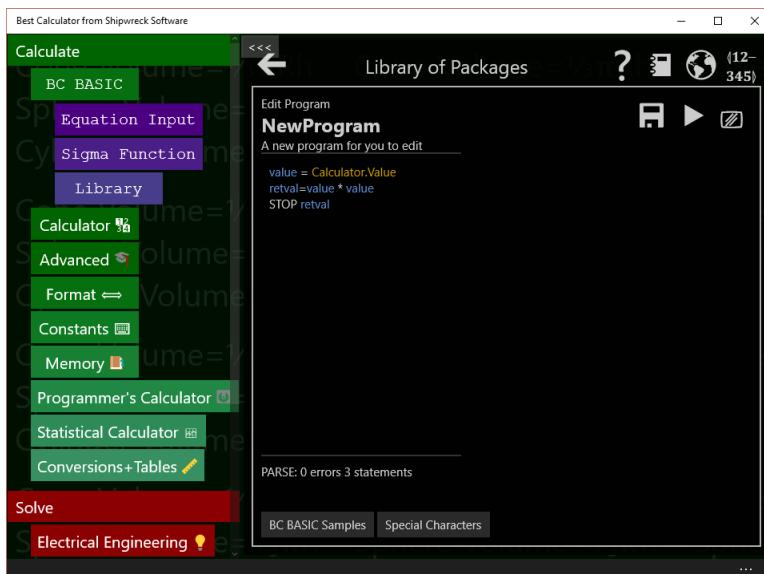
Change the name to “Square Feet to Acres” and the description to “Converts the current value in the calculator from square feet to acres”. You don’t have to do anything special to save the name and description; they are saved automatically when your program is saved.

You can also set the key label value. If you bind a program to a programmable key, this string will be displayed. Set it now to “FT>ACRE”. The string has to be short to fit onto a key.

### 38.3 EDIT THE PROGRAM TO DO THE CONVERSION

To get to the edit screen, you can either tap the EDIT key at the bottom of the About this program screen, or you can tap the BACK ARROW key to get back to the Programs list and then tap the program’s EDIT key.

The program starts out with a sort of mini-sample. You'll be deleting the mini-sample code and replacing it with your own.



Most conversion programs follow the same pattern:

1. Get a number from the calculator display. This is the number of square feet.
2. Multiple or divide it to get the new value. To convert square feet to acres, just divide by 43560.
3. Output a string to the calculator display to say what we've done
4. Return the numeric value so it's set into the calculator

The program to do these is:

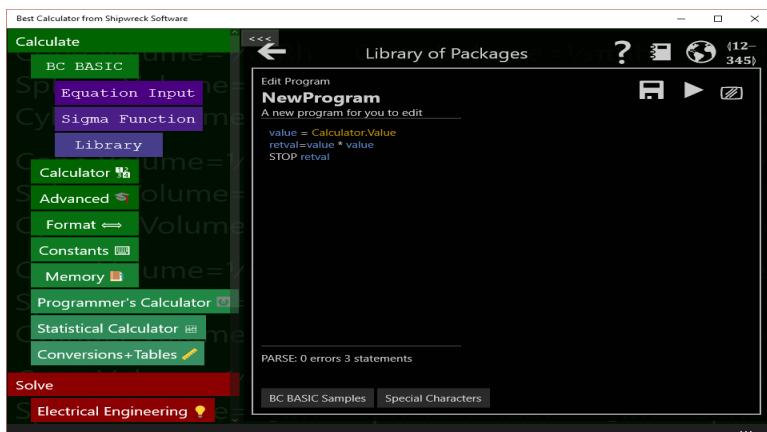
```
value = Calculator.Value  
newValue = value / 43560  
Calculator.Message = "Converted " + value ↓  
+ " square feet to acres"  
STOP newValue
```

Each line corresponds to one of our steps. Enter this code into the program area.

### 38.4 RUN THE PROGRAM TO TEST IT

Before we run the program, we need to have a “known good” conversion. Type “Convert 10000 square feet to acres” into a web search; it should tell you that 10000 square feet is 0.229568 acres.

To test the program, tap the Calculator key; this dismisses the BC BASIC programming area and pops up the calculator. Type in the starting value of 10000.



Now tap the BC BASIC key. The BC BASIC programming area pops up again, right where you were. Tap the RUN key to run the program. Your program is automatically saved when you run the program. You can also press the F5 key to run your program.

When you run the program from the editor, it will show a dialog box with the results. It's not displayed when you bind the program to a key. Tap the Calculator key again to see the calculator. You should see this:

The program works! It's done the conversion, and reminded you of



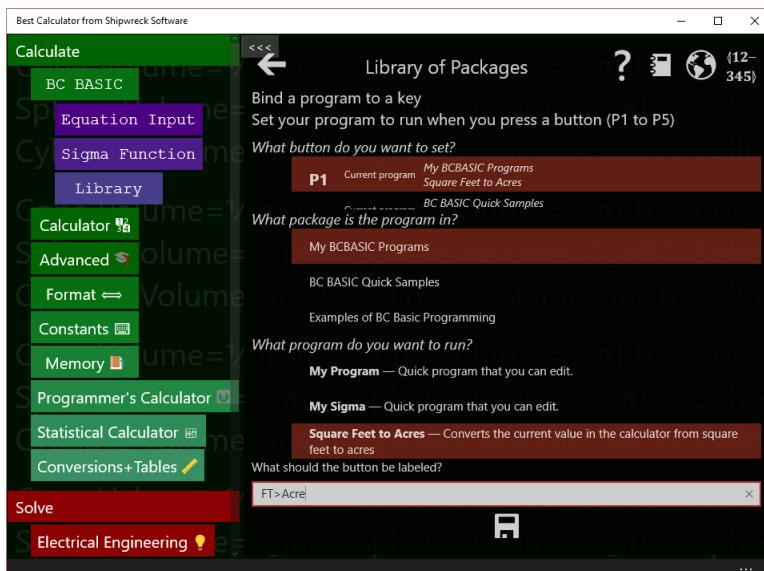
what exactly it did.

## 38.5 BIND THE PROGRAM TO A KEY

Now we're going to *bind* the program to one of the programmable keys on the calculator. They are the ones marked P1 to P4. Best Calculator comes with these keys already programmed to some common tasks.

In any of the programming dialogs, tap the BIND key ( 345); it's the one in the upper-right corner. It brings up the binding dialog.

To bind a key, you first pick the key to bind, the package and program to bind to, and then press save. You can also set what the key should say.



The first question is *What key do you want to bind to?* Tap one of the keys in the key list (labeled P1, P2, P3 and so on) to pick a key to bind to. People often just pick key P1. The key list tells you what package and program the key is currently bound to. This helps you pick the right key to use.

The second question is *What package is the program in?* All the possible packages are listed. As you tap on a package, the next list changes to show the programs in that package. Tap on **New Package** to pick your new package. If you've change the name of the package, pick the new name.

The third question is *What program do you want to run?* Tap **Square feet to acres** to select your new program.

You can optionally set the label of the key. The default value is set from the “Key Label” value when you update the program data (the “About this program” screen). You should have already set it to FT>ACRE.

Lastly, **be sure to tap the SAVE key** (). Your selection isn’t saved until you press save.

Now verify that the key works how it should. Tap the Calculator key to see the calculator again. Now enter a value into the calculator. You might want to enter 10000 since you already know how many acres it is. Now press the key you bound (probably the P1 key). The value in the calculator screen should be replaced with 0.229568 (and there is a message that it just converted 10000 square feet to acres).

## 38.6 NEXT STEPS

Now you’ve seen the dialogs and screens that you need to use to create a BC BASIC program. Your next step is to try it! Pick a problem that you have where you work, at home, for your hobby, or your schoolwork. Conversion programs are often a great way to start; lots of times you have to convert one value to another.

If you need to make a conversion program, take a look at the code in the Astronomy package. It demonstrates a more advanced way how to make a single central library program that handles lots of conversions. Or you can just write each one just like you did this one.

Sometimes you have to enter several numbers. The Arc Length program shows how you can prompt the user for several values. The Money Conversion program in the Quick Samples library shows how you can ask the user for input and remember the last value entered. By setting a default value for the value to be entered, you can really make your work flow go faster.

## 39 APPENDIX: RELEASE NOTES

---

Best Calculator is constantly updated. Starting in September 2017 the releases are marked with numbers; you can find out your current release number with **PRINT System.Version**

### 39.1 RELEASE NOTES: 3.20 MAY 2019

The Best Calculator, IOT edition was updated with additional Skoobot commands to control the Light and Distance sensors. New sample programs were added, and the existing programs updated to display light and distance values.

### 39.2 RELEASE NOTES: 3.19 MAY 2019

Two new features were added, plus in the BC BASIC console you can select even smaller fonts (down to size 6)

#### 39.2.1 Screen.GX and Screen.GY graphics sizes

You can now learn the graphics size of the text output screen. The size of the text output screen depends on the selected font size and the number of columns and rows.

#### 39.2.2 Skoobot robot support

The Skoobot is a tiny (1-inch cube) robot from  
<https://www.william-weiler-engineering.com/>

My Skoobot is red and has a slightly different shape on the laser-cut panels.



### 39.3 RELEASE NOTES: 3.18 APRIL 2019

Another large update with some powerful new features

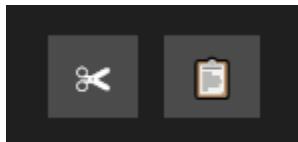
#### 39.3.1 IOT version is now free!

Best Calculator, IOT version is now fully free! It's free to download, with no ads, and with no in-app purchases! Enjoy!

This version also supports more down-level Windows 10 phones like the Acer Jade phone that's less than \$150 on Amazon!

### 39.3.2 Calculator COPY and PASTE

The calculator now supports both COPY and PASTE in the menu bar at the bottom of the screen. The scissors icon is for COPY and the CLIPBOARD (which used to be copy) is PASTE.



### 39.3.3 BC BASIC UI Import and Copy improvements

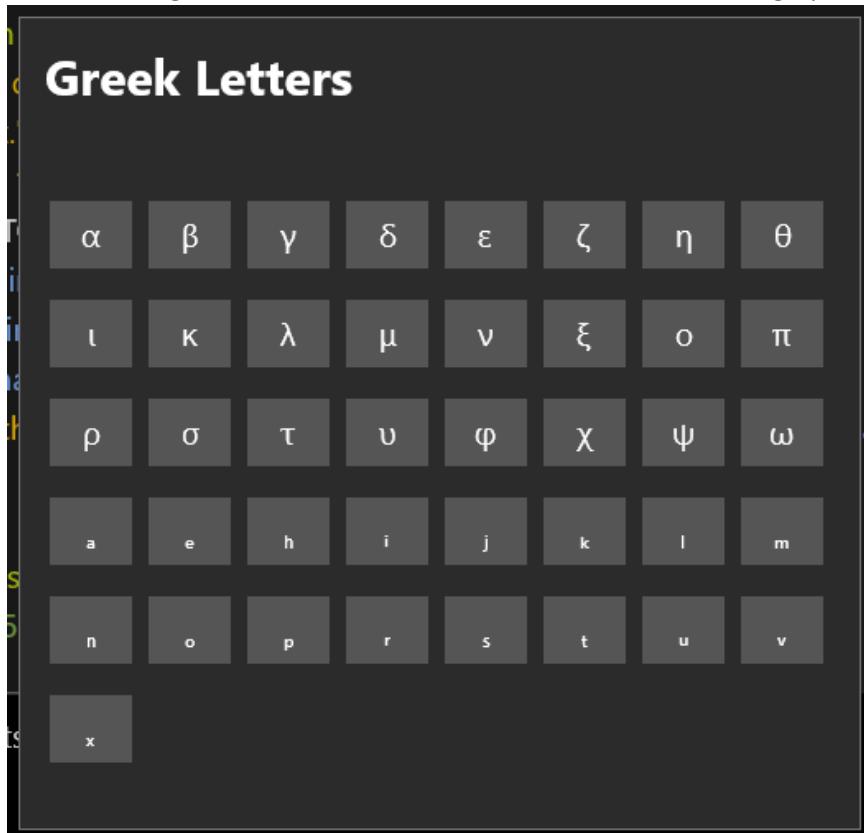
In BC BASIC, when you IMPORT a file, the new file is placed at the top of the list of packages not at the bottom. When there were just a few pre-made packages, this didn't matter so much; now there are over 25 packages that come with the IOT edition!

It's now easy to copy a BC BASIC locked system package to a user package that you can edit. Click the lock icon to show the system package properties and options, and then click the Unlock icon to unlock the package. You'll be told the name of the new package; it will start off being placed at the top of the list so you can find it easily.

### 39.3.4 Variable names can include more characters

Variable and function names can now include a selection of Greek and Coptic letters and a selection of subscripts. Names can start with Greek and Coptic letters, and the rest of the name can include Greek and Coptic letters and the selection of subscripts.

Available letters are listed in the Specialized menu.



You can now make an array with named values using the `Array SetProperty(name, value)` method. The values then be read or changed using a “dot” notation.

```
REM Create a data variable with fields  
DIM data()  
data SetProperty ("declination", 34)  
data SetProperty ("obliquity", .23)
```

```
REM Now print out the information  
CLS  
PRINT "Get the data declination"  
PRINT "declination", data.declination
```

### 39.3.5 New Data object

BC BASIC now includes a Data object that, over time, will have useful tables of information. In this version, you can get location information for a large number of cities (population >= 100000) over the entire planet.

#### 39.3.5.1 *Data location values*

The Data object includes a summary of place names from around the world. Cities must have a population of at least 100,000 people to be included in the list.

See the Data extension for more information.

### 39.3.6 DateTime improvements

#### 39.3.7 `DateTime.Parse (string)`

Makes a datetime from the given string

#### 39.3.8 `DateTime.Add (years, months, days, hours, minutes, seconds)`

Adds the amounts into a datetime

#### 39.3.9 `DateTime.DayOfYear`

For any datetime object, return the day number. For example, January 1<sup>st</sup> is 1, January 31<sup>st</sup> is 31, February 1 is 32. The day number will automatically adjust based on the leap year.

#### 39.3.10 `DateTime.HourDecimal`

Returns the current hour as a double, including the minutes in a decimal form. For example, if the current time is exactly 2:30 in the afternoon (14:30), the HoursDecimal will be 14.5.

#### 39.3.11 `DateTime.Set (years, months, days, hours, minutes, seconds)`

Creates a new DateTime object with the given values

#### 39.3.12 New Gopher object to make a Gopher of Things server

Gopher is a straightforward protocol that lets you design simple internet menu systems. The BC BASIC Gopher object lets you make a Gopher server that can display data and menus to remote users. The remote users will use a *Gopher Client* program like the [Simple Gopher Client](#) for Windows 10 computers and phones.

There's a full example of Gopher menus in the appendices.

### 39.3.13 Graphics.ClearGoTo

Clears the current position used by GoTo and LineTo. Use the ClearGoTo to draw multiple lines with LineTo and GoTo.

### 39.3.14 Graphics.YAxisMax and Graphics.YAxisMin

For Screen.GraphXY graphs, the Y axis normally automatically resizes to fit the data. You can override the minimum and maximum values with the graph.YAxisMin and graph.YAxisMax values.

## 39.4 RELEASE NOTES (3.17, SEPTEMBER 2018)

The September 2018 update is massive. Apart from everything else, the compiler and editor have been completely rewritten and are much faster (especially with large programs)

The Editor has changed dramatically! Users may know that the editor used to be OK for short programs, but had trouble keeping up when a large program is present. An entirely new editor has been created for BC BASIC. You will note that the speed is greatly enhanced and there are powerful new capabilities:

- There's a FIND and GO TO LINE ability
- You can INDENT code easily

In addition, the underlying “compiler” has been completely rewritten. It's now very substantially faster (the 2,000-line Wumpus game used to tax it severely; the same game compiles almost instantly in the new compiler). Although much effort was spent to ensure that programs compile and work the same way, it's likely that there will be a few changes that snuck through. Please accept my apologies in advance if this cause you any problems.

There are a TON more example programs including some fun game and math programs.

Setting colors is more consistent; color=<int> works for everything.

The INPUT expression used to require zero or one for DEFAULT and PROMPT, and they had to be in the order DEFAULT and then PROMPT. This is no longer required.

The PLAY command will PLAY STOP more quickly. In the past, PLAY STOP would wait until the current note stopped; now the note is stopped almost right away. This was important for the Hunt the Wumpus game, where the “pit” and “monster” sounds are drawn-out single notes (e.g., long ‘ocean’ notes). This prevented the user from making a move right away; the sound effects had to complete before the note could be played.

Arrays can be two-dimensional now! You can either make a 2-D array from the start with DIM or you can build one up.

Example making a 2-D array with DIM

```
REM Make an array with 2 rows and 5 cols
DIM array(2,5)
array.Fill (0)
array[r,c] = 1
PRINT array[r,c]
```

Example making a 2-D array with AddRow

```
REM Make an array with 2 rows and 5 cols
DIM array()
array.AddRow (11,12,13,14,15)
array.AddRow(21, 22, 23, 24, 25)
PRINT array[r,c]
```

The array.Add(<value>) method can now take multiple values. A call like **array.Add (1,2,3)** is just like calling

```
Array.Add (1, 2, 3)
```

```
REM The old way is more work!
```

```
array.Add (1)  
array.Add (2)  
array.Add (3)
```

There's a new array.Fill ( value) method that will entirely fill every element of a 1- or 2-dimensional array with the same value. The most common use is to zero-fill an entire array.

**Common example:**

```
DIM array(2,5)  
array.Fill (0.0)
```

INPUT can now read in multiple values at once. **INPUT X, Y, Z** will read three items from the user into the X, Y and Z variables.

READ can now read in multiple values at once. **READ X, Y, Z** will read three DATA items into the X, Y and Z variables.

Added RESTORE to reset the READ/DATA statements. READ normally starts at the first item in the first DATA statement and each subsequent READ picks the next item. RESTORE sets READ so that it will read from the first item again.

Added RND() as a function in addition to the normal RND value. This makes BC BASIC more compatible with other BASIC versions. RND() with no parameters will return a random number between 0 and 1, as will RND(positive number). The arguments isn't used. RND(0) will reset the random number generator in a random way, and RND(negative number) will reset the random number generator using the given seed.

Added SPC(n) function to improve compatibility with other version of BASIC. The SPC(n) function will return a string with <n> spaces (rounded down if needed). It's often used in conjunction with PRINT as in **PRINT SPC(5); "Hello"** to indent the Hello.

Added graphic LineTo(x,y) and GoTo(x,y). These make drawing some types of graphs much easier because your code doesn't have to keep track of the old x,y position.

The graphics g.Circle (x, y, radius) now has a get and set Radius property to change the radius of a circle.

Graphics now includes a set of Scaling methods to create a window on the screen, set the scale of the window, and to use different scaling modes. This is commonly used when creating physical-based displays: you can draw and move your graphs based on physical units like “centimeters” and have them automatically scale correctly. Look at the Bubble Animation demo for a good use of this ability.

Added bitwise operators for the Math object: Math.BitAnd (left, right), Math.BitOr (left, right) and Math.BitNot (value). These do bitwise operations. For example, PRINT Math.BitAnd (0x6, 0x3) will print 2. 0x6 is bit 0110 and 0x3 is 0011. Their AND is 0010.

The String.Parse (“csv”, “1,2,3”) method used to return only string type values. It will now attempt to convert each input to a double; if the conversion works it’s kept. The old way was a surprise especially when comparing the results; if data[1] has an actual value of 7, **IF (data[1] > 200) THEN PRINT “BIG”** would actually print BIG! That’s because BC BASIC would decide that the compare should be done as strings, and the string “7” is bigger than the string “200”. This cost me five wasted hours of debugging during a recent Hackathon!

Added String.Pos (string, lookFor, startingIndex) to search for a value inside a string starting at a given starting index.

Added String.Replace (string, startingIndex, length, replacement). The portions of the input string from startingIndex for the given length will be replaced with the given replacement. The size of the string to be replaced is entirely unrelated to the string replacement; you can “squeeze” a string to be smaller, or you can “expand” a string. The length to be replaced can have a length of zero, in which case the replacement string is spliced in.

This method is provided to make it easier to port programs from systems like the Tektronix 4050 series.

System.FolderBasic returns the folder that BASIC apps are saved to. System.FolderTemporary is a folder where time-stamped BASIC programs are saved.

System.Trace(0) works now; it had accidentally been disabled.

System.Trace(1) will cause some level of tracing in your program (more usefully, it says how many callbacks were suppressed and the number of lines/second are being evaluated). System.Trace(0) (the default) should suppress all tracing.

The TI SensorTag 1350 (and other TI Sensor Tags) are now manufactured without an IR temperature chip. TI's [website](#) says that they are no longer in the contactless IR sensor business, and no longer have any TMP007 in stock. The new boards have a blank solder pad where the chips would be. This condition is not readily detectable; you can still try to enable the IR Sensor, but you will not get any IR callbacks.

The Unicode list has been updated to Unicode version 11. New characters include OVERLAPPING WHITE AND BLACK SQUARES, the COPYLEFT SYMBOL, LAB COAT and SUPERVILLAIN.

### **Important bugs fixed:**

It's perfectly OK to make a nested (recursive) array. But in older version of BC BASIC, this would cause an instant crash in BC BASIC! The array code has been fixed; now when an array is printed and one of the values includes a sub-array, the sub-array printing is short-circuited,. The length of the sub-array is printed in superscripted number.

The test for this fix looks like the following:

```
FUNCTION TEST_NestedRecursiveArray()
  DIM a(3)
  a(1) = 1.1
  a(3) = 3.3
  a[2] = a
  string = ""+a
  ASSERT (string = "[1.1,[^],3.3]")
  RETURN
```

When the array 'a' is converted to a string, it's converted as [1.1,[^],3.3]. The array in the middle is replaced with [^], indicating that it's an array that's three elements long.

Other bugs fixed:

The CLS command didn't accept two numeric color values (like **CLS 1 2**). This now works.

## 39.5 RELEASE NOTES (3.15, FEBRUARY 2018)

Can get the CX and CY values from rectangular objects.

Text now includes **FontSize** as a set/get property, and you get get the **Text** value. Setting g.Text to a string will replace each “\n” in the string with a real new-line. This lets to make multi-line text boxes on the screen.

Graphics supports events for pointer **Pressed**, **Moved** and **Released** events. When you call g.**SetPressed** (“functionName”, “arg”) and then tap the screen, the function you specified will be called arguments (g, x, y, “arg”) where “arg” is whatever value you passed in originally. There are similar functions for g.**SetReleased**(“functionName”, “arg”) and g.**SetMoved** (“functionName”, “arg”). You will usually use different functions for each of these.

Rectangular objects have a **CX** and **CY** for the object’s X and Y center. You can set and get these values. You can also rotate an object obj.**Rotate** = radians where 0 is to the right and positive values rotate counter-clockwise (like they go in your geometry class). Can also set CXD and CYD which are the CX and CY Offsets. This value is a portion of the width and height and is normally .5

## 39.6 RELEASE NOTES (3.14, DECEMBER 2017)

It’s super easy to write and run **TEST** functions with **ASSERT()**s in your code. Add TEST functions to validate the functions that you make.

Added support for **READ** and **DATA** statements.

READ m will read data from the DATA statements into variable m. DATA 1,2,3,4 will create a set of data. Values in a data statement can be numbers or strings but not variables or expressions. READ and DATA are global; when a program starts all DATA statements will be found and added to the overall program state; as each READ happens (either in the global area or in a function) one item is taken from the global DATA statement.

Added support for the **INKEY\$** variable. Example: `a=INKEY$` will either return a blank string ("") or a keyboard key. The **INKEY\$** value will return the current keyboard input that hasn't been read so far.

To help with compatibility, the **ASC** function has been added as a synonym of CODE.

Added graphic.**Arc** (cx, cy, innerR, outerR, ang1, ang2) which draws an arc with an inner and outer radius from ang1 to ang2 (both in radians)

Added poly = graphic.**Polygon()** and the poly.**SetPoints**(x1, y1, x2, y2...) method. Call SetPoints() with the list of points that the polygon is for.

Added support for g.**Border** so you can control the color of a graphics window border. Also updated the code so that when you make a graphics window, it starts off as the color of the main fixed-size screen.

There is much better support for infinity. The  $\infty$  symbol is recognized as infinity and  $\infty$  has been added to the list of special symbols. Unlike Math.NaN,  $\infty$  can be compared in an expression. Lastly, **Math.Infinity** has been added for people who prefer to not type in non-ASCII characters.

The  $\not\equiv$  (NEITHER APPROXIMATELY EQUAL NOR ACTUALLY EQUAL TO) symbol has been added as a counterpoint to the approximately equal to sign.

Added support for the **Math.Sigma** (x) function. This function returns  $1 / (1 + e^{-x})$ ; mathematically, it maps an input which ranges anywhere from  $-\infty$  to  $+\infty$  into the range 0 to 1. This is often used with AI functions in order to squeeze data with an arbitrary range into a common range.

## 39.7 RELEASE NOTES (3.13, NOVEMBER 2017)

At the request of an IOT customer, button press callbacks from TI SensorTag device (all versions) will no longer be suppressed. Normally excessive callbacks (including Bluetooth data callbacks) are automatically suppressed so that the system doesn't get too laggy.

The original algorithm to suppress callbacks was that when it came time to call all pending callbacks, only the most recent call to a function

would be allowed; the rest would be discarded. The new algorithm is the same except that Button callback are except from triggering the discard algorithm.

## 39.8 RELEASE NOTES (3.12, NOVEMBER 2017)

Added full-screen graphics with `g = Screen.FullScreenGraphics()`. This method will create a graphics screen that is independent of the normal screen. It will cover the entire non-menu area of the calculator. A small “X” at the upper-right corner, when tapped, will hide the screen for about five seconds, allowing you to examine the regular screen or cancel the program. The full screen graphics screen will be removed automatically when the program stops.

Added graphics.Text (x1, y1, x2, y2, text, size) to display text. The text can be Unicode characters like “”.

Text includes a text.Align = “[LCRS][TCBS]” to set alignment. The first char sets the left/right alignment as Left, Center, Right or Stretch, and the second char set the up/down alignment as Top, Center, Bottom or Stretch.

Graphics shape objects (Circle/Ellipse, Rectangle, Text) can have their opacity set. For example, `rect.Opacity = 0.1`

The obj1.Intersect(obj2) method, which was added earlier, is documented in this release.

`g.Clear()` and `array.Clear()` will remove all object from a graphics screen or an array.

Improved the PRINT statement so that “orphan” semicolons and commas at the end of a PRINT statement will control what happens on the next statement. This is more compatible with other BASICs.

Example:

```
PRINT "ABC" ; "DEF" ;
PRINT "GHI"
```

This program used to print on two lines (ABCDEF on one and GHI on the other). Now it correctly print ABCDEFGHI all on one line.

## 39.9 RELEASE NOTES (3.11, NOVEMBER 2017)

Added g.**Button** and g.**Slider**, both of which return an object. Each has an Change method, a Text value. Slider also includes Min and Max.

**String.Parse** ("csv", array), if given an array-of-arrays, will create a correct 2-D table.

You can create pretty-print output from a package in **Markdown** format (MD) now. Markdown is an increasingly popular file format for writing computer documentation. Note that not all apps implement the same markdown. In particular, the markdown generated by Best Calculator marks source code with block:

CORRECTION: **MID(str, index)** is the same as **MID (str, index, rest-of-string)**. The documentation used to say that the default was "1", not "the rest of the string".

BUGS: early return in a FOR loop inside a function used to fail (in several ways); this is now fixed. The Rfcomm devices were not properly released and therefore couldn't always be connected to a second time.

## 39.10 VERSION 3.10, OCTOBER 2017 RELEASE NOTES

New devices were added: **Ardudroid**, Infineon **DPS310** and the Slant Robotics **LittleBot**. These are controlled using the new **Rfcomm** (SPP) Bluetooth support.

Each Array object now includes a **Mean** field like the SumOfSquares field.

The Rectangle object (from g.Rectangle()) now includes a **Data** member. This is for your use; it will never be examined or changed by BC BASIC. It's for when you need to save a little bit of information about a graphics object.

The Bluetooth device list methods (Devices, DevicesName and PickDevicesName) all have an **Rfcomm** version (DevicesRfcomm, DevicesRfcommName, PickDevicesRfcommName). The Rfcomm version will list Bluetooth devices that support the RF COMM (Serial port) protocols. Many Arduino devices, for example, use serial-port Bluetooth chips for communicating with computers.

Sensor.**Camera** and the graphics.**Image** let you grab the camera and display and analyze the image

## 39.11 VERSION 3.8, SEPTEMBER 2017 RELEASE NOTES

Pasting from PDF files and Word documents pastes only the text, not the formatting.

The **Graphics** object now includes g.**H** and g.**W** which return the size of the graphics windows in pixels.

The graphics.**Rectangle** object is now a full object. You can animate a rectangle by setting the X1 and Y1 properties.

Added a **System** object with useful system-wide properties including

- **System.Errors** is a list of the most recent errors. PRINT System.Errors and CONSOLE System.Errors are some of the easy ways to see the current set of errors. The error data persists between program runs, but not between restarts of the app.
- **System.SetInteval** (<function>, <delayInMilliseconds>, <argument>) calls function periodically.
- **System.Trace** (0) turns off tracing, System.Trace(1) turns on minimal tracing
- **System.Version** is the current version

The Sensor object was updated with Sensor.**Compass**

Sensor.**Inclinometer** and Sensor.**Light**. These are in addition to the existing Sensor.Location and Sensor.Microphone.

Multiple callbacks are more robust. All callbacks (like from Bluetooth devices) are serialized and will be called between regular statements, during PAUSE statements and during FOREVER statements.

## 39.12 AUGUST 2017, SECOND VERSION RELEASE NOTES

Added **BEEP** command to beep the computer speaker

Added **PLAY** command to play music and sounds. The simplest program that uses PLAY is **PLAY "C"** followed by **PLAY WAIT**. (If you don't do the

**PLAY WAIT**, the music will start for a fraction of a second and then stop when the program stop).

Added **SPEAK** “hello!” that will speak out loud. You can use any of the voices that are shipped with your system.

Added **FOREVER** (and the variant **FOREVER STOP**) to create an infinite loop that will allow callbacks (like for IOT events) and background updates (like the automatic graphs) to work.

Added `Calculator.ValueString` which returns the exact set of digits as presented in the calculator window. The regular `Calculator.Value` will return the same value but with full precision.

**TI 2541, 1350 and 2650** were documented as returning (device, ambient, object). The correct documentation is that the values are (device, object, ambient). The documentation has been updated to match the code.

The UNICODE dictionary has been updated to the most recent version. Newly added signs include POWER ON, POWER OFF, MAN DANCING and ZOMBIE.

### 39.13 AUGUST 2017 RELEASE NOTES

Added `Math.Fft(array)` and `Math.InverseFft (array)` to perform basic Fast Fourier Transforms on data.

Added `array.SumSquared`

Added **SQRT** as a synonym for **SQR**.

Sensor object includes machine sensors **Microphone** and **Location**

## 39.14 JULY 2017 RELEASE NOTES

The Statistics Calculator will now accept timestamps as data; the values will be converted into seconds.

Added the **Bluetooth.Watch**(watchFor, functionName) method to the Bluetooth object; this method will call functionName when specified Bluetooth devices are detected. All Bluetooth functions require the IOT edition.

There are new **Bluetooth** specializations for the **puck.js** and the **RuubiTag** (supported through **Bluetooth.Watch**). The Texas Instruments (TI) **Display (Watch)** DevPack for the TI SensorTag 1350 and 2650 are also supported now along with the **TI SensorTag 2650**.

Added **Screen.ClearLines**(startRow, endRow) to clear multiple lines on the screen.

The **Screen.Graphics()** method can now take 0 arguments (like before), or 2 arguments which are the X, Y position to display at, or 4 arguments which are the X, Y position plus the height and width. This lets you create, size and position a graphics object in one call instead of three.

**Strings** can now include embedded quotes: "this is ""quoted""!"

## 39.15 SPRING 2017 RELEASE NOTES

New in the Spring 2017 version of Best Calculator and Best Calculator, IOT Version

You can now place FOR ... NEXT loops inside of compound IF statements. Before these loops would not work correctly.

The **Array** object (made via a DIM statement) now includes new methods for making one and two-dimensional arrays and for dealing with data sampling.

- The AddRow method is an easy way to make an array-of-array; it helps create valid JSON strings from data.
- The Add() method adds to an array but will sample (or not) based on the MaxCount and RemoveAlgorithm values

There is a new **DateTime** extension that lets you create time stamps. This is especially useful when creating data from IOT sensors.

There is a new **File** extension that lets you read and write files. This is only available in the IOT edition.

There is a new **Screen.Graphics()** that lets you create draw lines, rectangles and circles on the screen. It includes a simple automatic graph capability for quickly making data graphs.

There is a new **Html** extension that lets you read and write data to the Internet. This is only available in the IOT edition.

The **Math.Round()** function can now take two parameters. The second parameter say how many decimal places to produce a result at.

The **Memory** extension now lets you save and restore string

There is a new **String** extension to parse and escape strings in common Internet formats like CSV and JSON.

There are new **Bluetooth** specializations for the BBC MicroBit, Mbient Labs devices and the TI SensorTag 1350.

## 39.16 RELEASE NOTES (3.15, JANUARY 2018)

Can get the CX and CY values from rectangular objects.

Text now includes FontSize as a set/get property, and you get get the Text value. Setting g.Text to a string will replace each “\n” in the string with a real new-line. This lets to make multi-line text boxes on the screen.

Graphics supports events for pointer Pressed, Moved and Released. When you call g.SetPressed (“functionName”, “arg”) and then tap the screen, the function you specified will be called arguments (g, x, y, “arg”) where “arg” is whatever value you passed in originally. There are similar functions for g.SetReleased(“functionName”, “arg”) and g.SetMoved (“functionName”, “arg”). You will usually use different functions for each of these.

Rectangular objects have a CX and CY for the object's X and Y center. You can set and get these values. You can also rotate an object obj.Rotate = radians where 0 is to the right and positive values rotate counter-clockwise (like they go in your geometry class). Can also set CXD and CYD which are the CX and CY Offsets. This value is a portion of the width and height and is normally .5

## 39.17 RELEASE NOTES (3.17, SEPTEMBER 2018)

The calculator now lets you pick from a couple of fonts including two antique digital fonts.

The Editor has changed dramatically! Users may know that the editor used to be OK for short programs, but had trouble keeping up when a large program is present. An entirely new editor has been created for BC BASIC. You will note that the speed is greatly enhanced and there are powerful new capabilities:

- There's a FIND and GO TO LINE ability
- You can INDENT code easily

In addition, the underlying "compiler" has been completely rewritten. It's now very substantially faster (the 2,000-line Wumpus game used to tax it severely; the same game compiles almost instantly in the new compiler). Although much effort was spent to ensure that programs compile and work the same way, it's likely that there will be a few changes that snuck through. Please accept my apologies in advance if this cause you any problems.

There are a TON more example programs including some fun game and math programs.

Setting colors is more consistent; color=<int> works for everything.

The INPUT expression used to require zero or one for DEFAULT and PROMPT, and they had to be in the order DEFAULT and then PROMPT. This is no longer required.

The PLAY command will PLAY STOP more quickly. In the past, PLAY STOP would wait until the current note stopped; now the note is stopped almost right away. This was important for the Hunt the Wumpus game, where the “pit” and “monster” sounds are drawn-out single notes (e.g., long ‘ocean’ notes). This prevented the user from making a move right away; the sound effects had to complete before the note could be played.

Arrays can be two-dimensional now! You can either make a 2-D array from the start with DIM or you can build one up.

Example making a 2-D array with DIM

```
REM Make an array with 2 rows and 5 cols
DIM array(2,5)
array.Fill (0)
array[r,c] = 1
PRINT array[r,c]
```

Example making a 2-D array with AddRow

```
REM Make an array with 2 rows and 5 cols
DIM array()
array.AddRow (11,12,13,14,15)
array.AddRow(21, 22, 23, 24, 25)
PRINT array[r,c]
```

The array.Add(<value>) method can now take multiple values. A call like `array.Add (1,2,3)` is just like calling

```
Array.Add (1, 2, 3)

REM The old way is more work!
array.Add (1)
array.Add (2)
array.Add (3)
```

There’s a new array.Fill ( value) method that will entirely fill every element of a 1- or 2-dimensional array with the same value. The most common use is to zero-fill an entire array.

Common example:  
`DIM array(2,5)`  
`array.Fill (0.0)`

`INPUT` can now read in multiple values at once. `INPUT X, Y, Z` will read three items from the user into the X, Y and Z variables.

`READ` can now read in multiple values at once. `READ X, Y, Z` will read three DATA items into the X, Y and Z variables.

Added `RESTORE` to reset the `READ/DATA` statements. `READ` normally starts at the first item in the first DATA statement and each subsequent `READ` picks the next item. `RESTORE` sets `READ` so that it will read from the first item again.

Added `RND()` as a function in addition to the normal `RND` value. This makes BC BASIC more compatible with other BASIC versions. `RND()` with no parameters will return a random number between 0 and 1, as will `RND(positive number)`. The arguments isn't used. `RND(0)` will reset the random number generator in a random way, and `RND(negative number)` will reset the random number generator using the given seed.

Added `SPC(n)` function to improve compatibility with other version of BASIC. The `SPC(n)` function will return a string with `<n>` spaces (rounded down if needed). It's often used in conjunction with `PRINT` as in `PRINT SPC(5); "Hello"` to indent the Hello.

Added graphic `LineTo(x,y)` and `GoTo(x,y)`. These make drawing some types of graphs much easier because your code doesn't have to keep track of the old x,y position.

The graphics `g.Circle (x, y, radius)` now has a get and set `Radius` property to change the radius of a circle.

Graphics now includes a set of Scaling methods to create a window on the screen, set the scale of the window, and to use different scaling modes. This is commonly used when creating physical-based displays: you can draw and move your graphs based on physical units like "centimeters" and have them automatically scale correctly. Look at the Bubble Animation demo for a good use of this ability.

Added bitwise operators for the Math object: Math.BitAnd (left, right), Math.BitOr (left, right) and Math.BitNot (value). These do bitwise operations. For example, PRINT Math.BitAnd (0x6, 0x3) will print 2. 0x6 is bit 0110 and 0x3 is 0011. Their AND is 0010.

The String.Parse ("csv", "1,2,3") method used to return only string type values. It will now attempt to convert each input to a double; if the conversion works it's kept. The old way was a surprise especially when comparing the results; if data[1] has an actual value of 7, **IF (data[1] > 200) THEN PRINT "BIG"** would actually print BIG! That's because BC BASIC would decide that the compare should be done as strings, and the string "7" is bigger than the string "200". This cost me five wasted hours of debugging during a recent Hackathon!

Added String.Pos (string, lookFor, startingIndex) to search for a value inside a string starting at a given starting index.

Added String.Replace (string, startingIndex, length, replacement). The portions of the input string from startingIndex for the given length will be replaced with the given replacement. The size of the string to be replaced is entirely unrelated to the string replacement; you can "squeeze" a string to be smaller, or you can "expand" a string. The length to be replaced can have a length of zero, in which case the replacement string is spliced in.

This method is provided to make it easier to port programs from systems like the Tektronix 4050 series.

System.FolderBasic returns the folder that BASIC apps are saved to. System.FolderTemporary is a folder where time-stamped BASIC programs are saved.

System.Trace(0) works now; it had accidentally been disabled. System.Trace(1) will cause some level of tracing in your program (more usefully, it says how many callbacks were suppressed and the number of lines/second are being evaluated). System.Trace(0) (the default) should suppress all tracing.

The TI SensorTag 1350 (and other TI Sensor Tags) are now manufactured without an IR temperature chip. TI's [website](#) says that

they are no longer in the contactless IR sensor business, and no longer have any TMP007 in stock. The new boards have a blank solder pad where the chips would be. This condition is not readily detectable; you can still try to enable the IR Sensor, but you will not get any IR callbacks.

The Unicode list has been updated to Unicode version 11. New characters include OVERLAPPING WHITE AND BLACK SQUARES , the COPYLEFT SYMBOL, LAB COAT and SUPERVILLAIN.

### Important bugs fixed:

It's perfectly OK to make a nested (recursive) array. But in older version of BC BASIC, this would cause an instant crash in BC BASIC! The array code has been fixed; now when an array is printed and one of the values includes a sub-array, the sub-array printing is short-circuited,. The length of the sub-array is printed in superscripted number.

The test for this fix looks like the following:

```
FUNCTION TEST_NestedRecursiveArray()
  DIM a(3)
  a(1) = 1.1
  a(3) = 3.3
  a[2] = a
  string = ""+a
  ASSERT (string = "[1.1,[^],3.3]")
RETURN
```

When the array 'a' is converted to a string, it's converted as [1.1,[<sup>3</sup>],3.3]. The array in the middle is replaced with [<sup>3</sup>], indicating that it's an array that's three elements long.

### Other bugs fixed:

The CLS command didn't accept two numeric color values (like **CLS 1 2**). This now works.

## 39.18 RELEASE NOTES (3.18, MAY 2019)

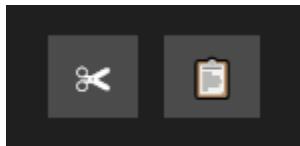
### 39.18.1 IOT version is now free!

Best Calculator, IOT version is now fully free! It's free to download, with no ads, and with no in-app purchases! Enjoy!

This version also supports more down-level Windows 10 phones like the Acer Jade phone that's less than \$150 on Amazon!

### 39.18.2 Calculator COPY and PASTE

The calculator now supports both COPY and PASTE in the menu bar at the bottom of the screen. The scissors icon is for COPY and the CLIPBOARD (which used to be copy) is PASTE.



### 39.18.3 BC BASIC UI Import and Copy improvements

In BC BASIC, when you IMPORT a file, the new file is placed at the top of the list of packages not at the bottom. When there were just a few pre-made packages, this didn't matter so much; now there are over 25 packages that come with the IOT edition!

It's now easy to copy a BC BASIC locked system package to a user package that you can edit. Click the lock icon to show the system package properties and options, and then click the Unlock icon to unlock the package. You'll be told the name of the new package; it will start off being placed at the top of the list so you can find it easily.

### 39.18.4 Variable names can include more characters

Variable and function names can now include a selection of Greek and Coptic letters and a selection of subscripts. Names can start with Greek and Coptic letters, and the rest of the name can include Greek and Coptic letters and the selection of subscripts.

### 39.18.5 Array SetProperty (name, value)

You can now create an array with named fields.

```
REM Create a data variable with fields  
DIM data()  
data SetProperty ("declination", 34)  
data SetProperty ("obliquity", .23)  
  
REM Now print out the information  
CLS  
PRINT "Get the data declination"  
PRINT "declination", data.declination
```

The resulting output is

```
Get the data declination  
declination      34
```

### 39.18.6 New Data object

BC BASIC now includes a Data object that, over time, will have useful tables of information.

#### *39.18.6.1 Data location values*

The Data object includes a summary of place names from around the world. Cities must have a population of at least 100,000 people to be included in the list.

```
location = Data.PickLocation()
```

Lets the user pick a location; returns an array with properties:

```
Location.Name  
Location.Latitude  
Location.Longitude
```

```
REM Get all the cities called york  
Locations = Data.GetLocations ("York")
```

#### 39.18.7 DateTime.Parse (string)

Makes a datetime from the given string

#### 39.18.8 DateTime.Add (years, months, days, hours, minutes, seconds)

Adds the amounts into a datetime

### 39.18.9 DateTime.DayOfYear

For any datetime object, return the day number. For example, January 1<sup>st</sup> is 1, January 31<sup>st</sup> is 31, February 1 is 32. The day number will automatically adjust based on the leap year.

### 39.18.10 DateTime.HourDecimal

Returns the current hour as a double, including the minutes in a decimal form. For example, if the current time is exactly 2:30 in the afternoon (14:30), the HoursDecimal will be 14.5.

### 39.18.11 DateTime.Set (years, months, days, hours, minutes, seconds)

Creates a new DateTime object with the given values

### 39.18.12 Graphics.ClearGoTo

Clears the current position used by GoTo and LineTo. Use the ClearGoTo to draw multiple lines with LineTo and GoTo.

### 39.18.13 Graphics.YAxisMax and Graphics.YAxisMin

For Screen.GraphXY graphs, the Y axis normally automatically resizes to fit the data. You can override the minimum and maximum values with the graph.YAxisMin and graph.YAxisMax values.

## 39.19 RELEASE NOTES (3.19, MAY 2021)

### 39.19.1 Screen.GX and Screen.GY

You can now get the graphic size of the terminal screen with the Screen.GX and Screen.GW values. These are useful to make a graphics display that's neatly laid out on the terminal screen.

In the example, text is placed on the left side of the screen and a graphical area is on the right. The graphics position includes the border, but the size does not.

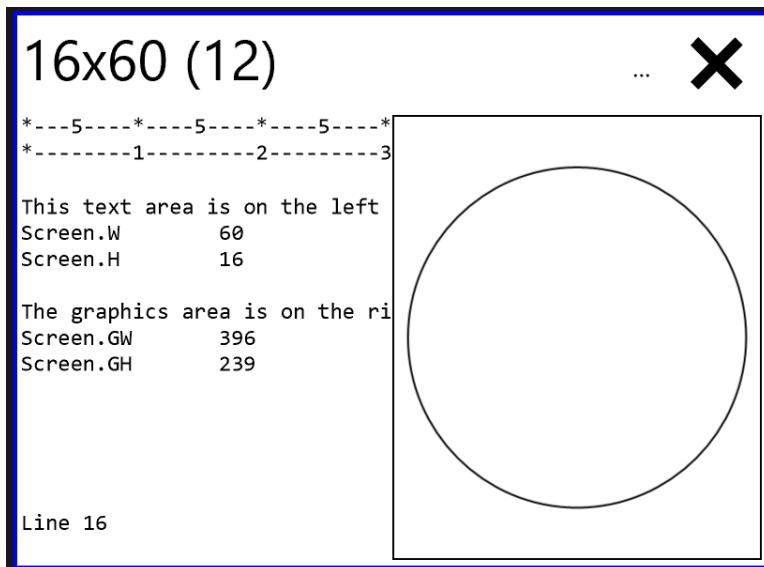
```
REM Make a small text area on the left
REM and a graphics area on the right

CLS WHITE BLACK
PRINT AT 1,1 "*----5----*----5----*----5----*----5"
PRINT AT 2,1 "-----1-----2-----3----"
PRINT AT 4,1 "This text area is on the left"
```

```
PRINT AT 5,1 "Screen.W", Screen.W
PRINT AT 6,1 "Screen.H", Screen.H
PRINT AT 8,1 "The graphics area is on the right"
PRINT AT 9,1 "Screen.GW", Screen.GW
PRINT AT 10,1 "Screen.GH", Screen.GH
PRINT AT 16, 1"Line 16"

REM The graphics screen is placed based on the
REM outer position of the graphics (including the
REM border area). The screen size is based on
REM the inner area.
REM Hence the need for account for the padding.
PADDING = 1
X1 = Screen.GW/2 - PADDING
Y1 = 0
W = Screen.GW - X1 - PADDING*2
H = Screen.GH - Y1 - PADDING*2
g= Screen.Graphics (X1, Y1, H, W)
g.Border = BLACK
g.Fill = WHITE
g.Circle (W/2, H/2, Math.Min (W/2, H/2)-5)
```

The output looks like this:



The output will always have the same proportions regardless of the characters size of the screen or the size of the font.

### 39.19.2 Skoobot is now supported!

The Skoobot is a tiny Bluetooth-controllable robot from <https://www.william-weiler-engineering.com/>. Sample programs include the simplest possible program to set the robot into Rover mode, a keyboard-driven robot program, and a GUI program.



Use `device = Bluetooth.PickDeviceName("Skoobot")` to let the user select one particular Skoobot. Then to get a Skoobot specialization, use the `device.As` method, passing in "Skoobot" like this: `skoobot = device.As ("Skoobot")`

Available Skoobot methods are

Category	Methods
Raw Motion Commands	<code>Left30()</code> <code>Right30()</code> <code>Forward()</code> <code>Backward()</code>
Stop	<code>Stop()</code>
Sounds	<code>PlayBuzzer()</code>
AI modes	<code>RoverMode()</code> <code>RoverModeRev()</code> <code>FotovoreMode()</code>

For example, to have the Skoobot move forward, once you have a `skoobot` specialization, just call `skoobot.Forward()`

These correspond directly to the underlying Bluetooth commands. The Skoobot primary commands are using service 00001523-1212-efde-1523-785feabcd123 and characteristic 00001525-1212-efde-1523-785feabcd123. You might be familiar with these as they are the standard GUIDs for the Nordic Semiconductor nRF Blinky app. The meaning of the command byte has been extended so to include the Skoobot commands

Skoobot Command	Command Value
-----------------	---------------

Right30	0x08
Left30	0x09
Right	0x10
Left	0x11
Forward	0x12
Backward	0x13
Stop	0x14
StopTurning	0x15
MotorsSleep	0x16
PlayBuzzer	0x17
RoverMode	0x40
FotovoreMode	0x41
RoverModeRev	0x42

The Skoobot specialization understands all these commands.

There are complete Skoobot sample programs including

**“A first control program”** which is the simplest possible real Skoobot program. It demonstrates how to connect to a Skoobot and to get a Skoobot specialization.

**“Control program for Skoobot”** is a GUI control program

**“Keyboard-driven Skoobot program”** is a command-line program for the Skoobot.

The first control program is:

```

CLS
PRINT "SKOOBOT CONTROL PROGRAM"
PRINT "Sets the Skoobot into Rover mode"
PRINT "Will automatically stop after 5 seconds"

device = Bluetooth.PickDevicesName ("Skoobot*")
IF (device.IsError) THEN
    PRINT "No device selected"
    STOP
END IF

REM get the specialization of the device
skoobot = device.As ("Skoobot")

```

```
skoobot.RoverMode()
```

```
REM Run for about 5 seconds  
PAUSE 50*5  
skoobot.Stop()
```

```
STOP
```

## 39.20 RELEASE NOTES (3.20, DECEMBER 2021)

### 39.20.1 Unicode tables are updated to the latest standard

The Unicode tables now reflects the latest 2021 Unicode set.

### 39.20.2 INKEY\$ is initialized correctly

Originally, the INKEY\$ value would possibly be set by the “F5” press used to start a program. This has been corrected.

### 39.20.3 STOP SILENT command

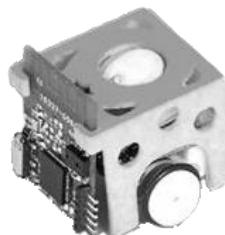
The STOP command now has a SILENT option; when added to STOP the program will stop running but a dialog won’t be displayed to the user. The dialog is awkward for some programs.

### 39.20.4 Gopher.Start(..., port)

The Gopher.Start method now takes an addition port number as a parameter. Normally the Gopher server is on port 70; you can set it to any other internet port number.

### 39.20.5 Skoobot updates!

The Skoobot is a tiny Bluetooth-controllable robot from <https://www.william-weiler-engineering.com/>. Sample programs include the simplest possible program to set the robot into Rover mode, a keyboard-driven robot program, and a GUI program.



New in this release: the Skoobot specialization can provide a stream of distance and light values. You just have to pass in the function you want called and the number of milliseconds between callbacks.

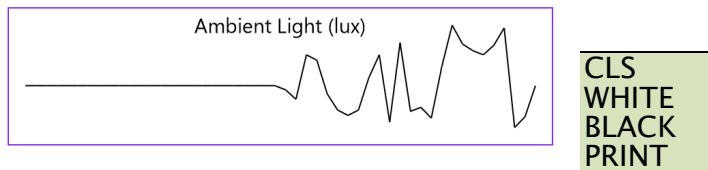
The light callback gets a value in Lux. The distance callback gets a distance in centimeters (cm).

Category	Methods
Data command	SetupLight("function", ms) SetupDistance("function", ms)

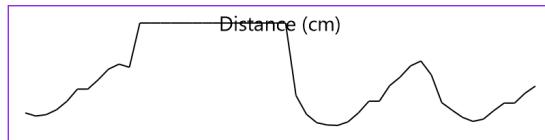
Example: complete program to graph Light data

This example is a complete program that demonstrates setting up an ambient light callback, adding data to an array, and automatically graphing that data.

```
SKOOBOT LIGHT AND DISTANCE PROGRAM
Ambient      64
Distance    14.478
```



```
CLS
WHITE
BLACK
PRINT
```



### "SKOOBOT LIGHT AND DISTANCE PROGRAM"

```

REM
REM Pick a Skoobot. If there's only one, select it
REM automatically with no user intervention.
REM
devices = Bluetooth.DevicesName("Skoobot*")
IF (devices.Count = 0)
    PRINT "No Skoobot devices found!"
END IF
IF (devices.Count = 1)
    device = devices[1]
ELSE
    device = Bluetooth.PickDevicesName
    ("Skoobot*")
    IF (device.IsError) THEN
        PRINT "No device selected"
        STOP
    END IF
END IF
skoobot = device.As ("Skoobot")
```

```
REM Set up a Light array
DIM lightData()
lightData.MaxCount = 50
lightData.RemoveAlgorithm = "First"
gl = Screen.Graphics (50, 50, 100, 400)
gl.Title = "Ambient Light (lux)"
gl.GraphY (lightData)

REM Set up a distance array
DIM distanceData()
distanceData.MaxCount = 50
distanceData.RemoveAlgorithm = "First"
gd = Screen.Graphics (50, 180, 100, 400)
gd.Title = "Distance (cm)"
gd.GraphY (distanceData)

REM
REM Set up the Light and Distance functions.
REM Each will be called back about every 100
milliseconds.
REM
skoobot = device.As ("Skoobot")
skoobot.SetupLight ("Light", 100)
skoobot.SetupDistance ("Distance", 100)
FOREVER WAIT

REM Called with light values in LUX
FUNCTION Light (currskoobot, lux)
    GLOBAL lightData
    lightData.Add (lux)
    Screen.ClearLine (2)
    PRINT "Ambient", lux
RETURN

REM Called with an approximate distance in cm
(centimeter)
FUNCTION Distance (currskoobot, cm)
    GLOBAL distanceData
    distanceData.Add (cm)
    Screen.ClearLine (3)
    PRINT "Distance", cm
RETURN
```

There are complete Skoobot sample programs including

**“A first control program”** which is the simplest possible real Skoobot program. It demonstrates how to connect to a Skoobot and to get a Skoobot specialization.

**“Control program for Skoobot”** is a GUI control program

**“Gopher-of-things for Skoobot”** shows how to control the Skoobot over the internet using the Gopher protocol to a local gateway. The local gateway runs the Gopher-of-things program, connecting to the nearby Skoobot using Bluetooth and offering up a Gopher interface to any Gopher client over the internet.

**“Keyboard-driven Skoobot program”** is a command-line program for the Skoobot.

**“Light and Distance”** demonstrates getting just light and distance data straight from the Skoobot and graphing the results. BC BASIC is particularly good at making it really simple to make graphs that automatically update themselves.

## 40 APPENDIX: SAMPLE PROGRAMS

---

This appendix includes many of the sample programs that come with Best Calculator and Best Calculator, IOT Edition. You can use these sample to help create your own program and understand how different Bluetooth devices are programmed.

All of the packages that start with BT: require the Bluetooth capabilities of Best Calculator, IOT Edition.

All of the packages that start with EX: work with all the BC Basic that's included in all recent Best Calculator editions.

### 40.1 BT: AN OVERVIEW OF BLUETOOTH

Introduces Bluetooth programming. Call the Bluetooth.Devices() method to get a list of paired Bluetooth devices for a system. For each individual device, you can get the name or you can make Bluetooth calls into individual devices. The device.Init() call is needed to get real Bluetooth device data.

#### 40.1.1 List Bluetooth devices

Call the Bluetooth.Devices() method to get a list of paired Bleutooth devices. For each device in the list you can get the name even without call the device.Init() method. The list.Count property is the way to get the length of the list.

```
CLS BLUE
PRINT "Available Bluetooth devices"

devices = Bluetooth.Devices ()

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    PRINT "NAME", device.Name
NEXT i
```

```
n = devices.Count  
PRINT ""  
PRINT "" + n + " devices were found"
```

### 40.1.2 Pick a Bluetooth device

The `Bluetooth.PickDevicesName(<name pattern>)` method lets the user select a single Bluetooth device from a matching list.

```
CLS BLUE  
PRINT "PickDevicesName lets the user select"  
PRINT "a single Bluetooth device from a list"  
PRINT ""  
device = Bluetooth.PickDevicesName("")  
IF (device.IsError)  
    PRINT "Sorry, no device was picked"  
ELSE  
    PRINT "Device ";device.Name;" was picked!"  
    PRINT device.Properties  
END IF
```

### 40.1.3 Power

Get real data from each Bluetooth device using the raw Bluetooth read commands. This program builds on the List program: each device is initialized with the `device.Init()` call. Once initialized, standard Power data is retrieved from each device. There are two types of reads: cached reads (like `device.ReadRawByte`) are faster because they use the data that the operating already knows. The raw reads will use the Bluetooth radio and will ask the device for data. Each raw call gets the most up to date data (but will be slower).

```
CLS BLUE  
PRINT "Read Bluetooth Power"  
  
REM  
REM How many Bluetooth devices are available?
```

```
REM
```

```
devices = Bluetooth.Devices ()
```

```
FOR i = 1 TO devices.Count
```

```
    device = devices.Get(i)
```

```
    PRINT "NAME", device.Name
```

```
    GetPowerInfo(device)
```

```
NEXT i
```

```
REM Get power data using the RAW bluetooth routines
```

```
FUNCTION GetPowerInfo(bt)
```

```
    PRINT "Init", bt.Init()
```

```
    PRINT "POWER", bt.ReadRawByte("180f", "2a19")
```

```
    PRINT "CACHE", bt.ReadCachedByte("180f", "2a19")
```

```
    PRINT "BLE_Name", bt.BLE_Name
```

```
RETURN
```

## 40.2 BT: BBC MICROBIT

Demonstrates how to use the BBC micro:bit device. The micro\_bit is a small, battery-powered computer, programmable in Python and other languages; it can be configured to send data over Bluetooth. The sensors include an accelerometer, magnetometer, temperature sensor. It also includes buttons for input, can control IO pins directly and has a 5x5 LED output that can be set as a bitmap or can have scrolling text.

### 40.2.1 Accelerometer

Demonstrates the basics of the AccelerometerSetup and using a callback routine. The callback routine will be called with the device and an X, Y and Z acceleration values. The units are in terms of G, where 1.0 is normal gravity.

```
CLS BLUE
```

```
PRINT AT 5,1 "Demonstrate micro:bit Accelerometer"
```

```
device = Bluetooth.PickDevicesName("BBC micro:bit")
```

```
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("microbit")
    PRINT AT 6,1 "Got a device", device.Name
    REM 1=turn on the on-device accelerometer
    REM 20=accelerometer update speed (in milliseconds)
    period = INPUT DEFAULT 100 PROMPT "Period (in millisecond) 1, 2,
5, 10, 20, 80, 160 and 640"
    PRINT AT 7,1 "SETUP", tag.AccelerometerSetup(1, period,
"Accelerometer")

    PRINT AT 8,1 "Done with setup"

    REM Now wait a little while. The Accelerometer routine will
    REM be called with updates.
    FOR time = 1 TO 10
        Screen.ClearLine(3)
        PRINT "TIME", time
        PAUSE 50
    NEXT time

    PRINT AT 9, 1 "FINISH", status
    tag.AccelerometerSetup(0, period, "Accelerometer")
END IF

FUNCTION Accelerometer(tag, x, y, z)
    Screen.ClearLine(1)
    PRINT x, y, z
RETURN
```

## 40.2.2 Button

The micro:bit includes two buttons, A and B. This program demonstrates how to set up a callback routine that will be called with the state of either the A or B button changes.

```
CLS BLUE
```

```
PRINT AT 5,1 "Demonstrate microbit Buttons"
```

```
PRINT AT 6,1 "Count", devices.Count
```

```
device = Bluetooth.PickDevicesName("BBC micro:bit")
```

```
IF (device.IsError)
```

```
    PRINT "No device was picked"
```

```
ELSE
```

```
    tag = device.As("microbit")
```

```
    PRINT AT 7,1 "SETUP", tag.ButtonSetup(1, "Button")
```

```
FOR time = 1 TO 30
```

```
    PAUSE 50
```

```
    PRINT AT 3,1 "TIME", time
```

```
NEXT time
```

```
    PRINT AT 8,1 "CLOSE", tag.ButtonSetup(0, "Button")
```

```
END IF
```

```
FUNCTION Button(tag, A, B)
```

```
    Screen.ClearLine(1)
```

```
    IF (A) THEN PRINT AT 1,1 "A"
```

```
    IF (B) THEN PRINT AT 1,8 "B"
```

```
RETURN
```

### 40.2.3 Compass

Demonstrates the basics of the CompassSetup and using a callback routine. The micro:bit, in addition to the raw magnetometer data also includes an easy way to get a magnetic compass heading. The callback will be called with the device and with the heading in degrees where 0 and 360 both mean magnetic north.

```
CLS BLUE
```

```
PRINT AT 5,1 "Demonstrate micro:bit Compass"
```

```
device = Bluetooth.PickDevicesName("BBC micro:bit")
```

```
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("microbit")
    PRINT AT 6,1 "Got a device", device.Name
    period = INPUT DEFAULT 100 PROMPT "Period (in millisecond) 1, 2,
5, 10, 20, 80, 160 and 640"
    PRINT AT 7,1 "SETUP", tag.CompassSetup(1, period, "Compass")

    PRINT AT 8,1 "Done with setup"

REM Now wait a little while. The Compass routine will
REM be called with updates.
FOR time = 1 TO 10
    Screen.ClearLine(3)
    PRINT "TIME", time
    PAUSE 50
NEXT time

PRINT AT 9, 1 "FINISH", status
tag.CompassSetup(0, period, "Compass")
END IF

FUNCTION Compass(tag, bearing)
    Screen.ClearLine(1)
    PRINT "bearing", bearing
RETURN
```

#### 40.2.4 Magnetometer

Demonstrates the basics of the MagnetometerSetup and using a callback routine.

```
CLS BLUE
```

```
PRINT AT 5,1 "Demonstrate micro:bit Magnetometer"
```

```
device = Bluetooth.PickDevicesName("BBC micro:bit")
```

```
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("microbit")
    PRINT AT 6,1 "Got a device", device.Name
    REM 1=turn on the on-device magnetometer
    period = INPUT DEFAULT 100 PROMPT "Period (in millisecond) 1, 2,
5, 10, 20, 80, 160 and 640"
    PRINT AT 7,1 "SETUP", tag.MagnetometerSetup(1, period,
"Magnetometer")

    PRINT AT 8,1 "Done with setup"

    REM Now wait a little while. The Magnetometer routine will
    REM be called with updates.
    FOR time = 1 TO 10
        Screen.ClearLine(3)
        PRINT "TIME", time
        PAUSE 50
    NEXT time

    PRINT AT 9, 1 "FINISH", status
    tag.MagnetometerSetup(0, period, "Magnetometer")
END IF

FUNCTION Magnetometer(tag, x, y, z)
    Screen.ClearLine(1)
    PRINT x, y, z
RETURN
```

## 40.2.5 SetLed

Demonstrates how to set the LED 'screen' of the device. The method takes 5 values, one for each row in the screen.

```
CLS BLUE
```

```
PRINT AT 5,1 "Demonstrate micro:bit Write (text, speed)"
```

```
device = Bluetooth.PickDevicesName("BBC micro:bit")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("microbit")
    PRINT AT 6,1 "Got a device", device.Name
    REM Draws a diagonal line. the top row has a couple
    REM more LEDs turned on.
    REM 0x07==3 bits, on the right, top row
    REM 0x02==second row, etc.
    PRINT "status", tag.SetLed (0x07, 0x02, 0x04, 0x08, 0x10)
END IF
```

## 40.2.6 Status

Shows how to get some basic information out of the device.

```
CLS BLUE
PRINT "An introduction to the Microbits specialization"

device = Bluetooth.PickDevicesName("*BBC*")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    PRINT "BBC Device"
    PRINT "Name", device.Name
    tag = device.As ("microbit")
    PRINT "Methods", tag.Methods
    PRINT "GetName()", tag.GetName()
    PRINT " "
END IF
```

## 40.2.7 Temperature

Demonstrates the TemperatureSetup method which sets up a callback function that will be called when the temperature data changes. The

callback function will be called with the device and the temperature in degrees Celsius.

```
CLS BLUE
PRINT AT 5,1 "Demonstrate micro:bit temperature"

device = Bluetooth.PickDevicesName("BBC micro:bit")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("microbit")
    PRINT AT 6,1 "Got a device", device.Name
    REM 1=turn on the on-device temperature sensor
    period = INPUT DEFAULT 100 PROMPT "Period (in millisecond) 1, 2,
5, 10, 20, 80, 160 and 640"
    PRINT AT 7,1 "SETUP", tag.TemperatureSetup(1, period,
"Temperature")

    PRINT AT 8,1 "Done with setup"

    REM Now wait a little while. The temperature routine will
    REM be called with updates.
    FOR time = 1 TO 10
        Screen.ClearLine(3)
        PRINT "TIME", time
        PAUSE 50
    NEXT time

    PRINT AT 9, 1 "FINISH", status
    tag.TemperatureSetup(0, period, "Temperature")
END IF

FUNCTION Temperature(tag, degreesC)
    Screen.ClearLine(1)
    PRINT "TEMP", degreesC
```

**RETURN**

### 40.2.8 Write (text, speed)

Demonstrates how to write text on the LED 'screen' of the device. The method takes two parameters: a string to display and a speed. The speed says how fast the text will scroll on the screen. A good value is 100.

CLS BLUE

PRINT AT 5,1 "Demonstrate micro:bit Write (text, speed)"

```
device = Bluetooth.PickDeviceName("BBC micro:bit")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("microbit")
    PRINT AT 6,1 "Got a device", device.Name
    text = INPUT DEFAULT "hello" PROMPT "Text to write"
    speed = INPUT DEFAULT 100 PROMPT "Scroll speed"
    PRINT "status", tag.Write (text, speed)
END IF
```

## 40.3 BT: BELIGHT

Supports the beLight CC2540T developer kit from TI. This is a small, Bluetooth-enabled high-output light. Unlike some other lights, it includes a bright white light plus individual red, green and blue lights. This lets you make a light whose color can be adjusted to be cooler (more blue) or warmer (more red). The BC BASIC device.As("beLight") specialization includes just one method, SetColor(r, g, b, w) that lets you set the red, green, blue and white values. Valid values are 0 (off) to 255.

### 40.3.1 Green

Turns the device green

```
CLS BLUE
```

```
PRINT "Sets the beLight to green"
```

```
device = Bluetooth.PickDevicesName("beLight")
```

```
IF (device.IsError)
```

```
    PRINT "No device was picked"
```

```
ELSE
```

```
    beLight = device.As ("beLight")
```

```
    REM The four parameters are Red, Green, Blue and White values.
```

```
    REM White is very bright
```

```
    REM They must be in the range 0 to 255
```

```
    Status = beLight.SetColor (0, 255, 0, 0)
```

```
    PRINT "status", Status
```

```
END IF
```

### 40.3.2 Red

Turns the beLight red

```
CLS BLUE
```

```
PRINT "Sets the beLight to red"
```

```
device = Bluetooth.PickDevicesName("beLight")
```

```
IF (device.IsError)
```

```
    PRINT "No device was picked"
```

```
ELSE
```

```
    beLight = device.As ("beLight")
```

```
    REM The four parameters are Red, Green, Blue and White values.
```

```
    REM White is very bright
```

```
    REM They must be in the range 0 to 255
```

```
    Status = beLight.SetColor (255, 0, 0, 0)
```

```
    PRINT "status", Status
```

```
END IF
```

## 40.4 BT: DOTTI

Demonstrates how to control the Dotti device (from Witti design company). The DOTTI device is a small desktop device with an 8x8 array of pixels. Each pixel can be programmed individually. The Pairing code is 123456.

### 40.4.1 An introduction

An introduction to using the DOTTI specialization

```
CLS BLUE
PRINT "An introduction to the DOTTI specialization"

device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    PRINT "Dotti Device"
    PRINT "Name", device.Name
    Dotti = device.As ("DOTTI")
    PRINT "Methods", Dotti.Methods
    PRINT "GetName()", Dotti.GetName()
    PRINT "GetPower()", Dotti.GetPower()
    PRINT " "
END IF
```

### 40.4.2 Change Mode

Changes the mode of the Dotti device (clock, animation, etc)

```
CLS BLUE
PRINT "Set DOTTI mode"
PRINT "0=default on"
PRINT "1=Animation"
PRINT "2=Clock"
PRINT "3=Dice Game"
PRINT "4=Battery Indicator"
```

```
PRINT "5=Off"
```

```
device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    device = devices.Get(i)
    Dotti = device.As ("DOTTI")
    mode = INPUT DEFAULT 1 PROMPT "What mode?"
    Status = Dotti.ChangeMode(mode)
    PRINT "status", Status
END IF
```

#### 40.4.3 List BT Devices

Lists all of the available paired Bluetooth devices and prints both the Windows version of the name and the BLE (Bluetooth device) version of the name. These can be different on DOTI devices: when you use the DOTI commands to change the name, the BLE name will change. But the Windows name might only change after restarting or re-pairing the device.

```
CLS BLUE
PRINT "Read Bluetooth Power"

REM
REM How many Bluetooth devices are available?
REM
devices = Bluetooth.Devices ()

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    PRINT "NAME", device.Name
    GetPowerInfo(device)
NEXT i

REM Get power data using the RAW bluetooth routines
```

```
FUNCTION GetPowerInfo(bt)
    PRINT "Init", bt.Init()
    PRINT "POWER", bt.ReadRawByte("180f", "2a19")
    PRINT "CACHE", bt.ReadCachedByte("180f", "2a19")
    PRINT "BLE_Name", bt.BLE_Name
RETURN
```

#### 40.4.4 Load screen from memory

Loads the screen from memory (animation, dice, notifications, etc)

```
Status = Dotti.ChangeMode(2)

CLS BLUE
PRINT "Load Screen"

device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    device = devices.Get(i)
    Dotti = device.As ("DOTTI")
    ez = INPUT DEFAULT 24 PROMPT "What icon?"
    Part1=EZValueToPart1(ez)
    Part2=EZValueToPart2(ez)
    PRINT "ez", ez
    PRINT "Part1", Part1
    PRINT "Part2", Part2
    Dotti.LoadScreenFromMemory(Part1, Part2)
END IF

FUNCTION ShowAllScreens(Dotti)
    FOR ez = 1 TO 9
        Part1=EZValueToPart1(ez)
        Part2=EZValueToPart2(ez)
        PRINT ez, Part1, Part2
        Dotti.LoadScreenFromMemory(Part1, Part2)
```

```
PAUSE 50
NEXT ez
RETURN

FUNCTION EZValueToPart1(ez)
IF (ez = 0) THEN RETURN 0
IF (ez < 9) THEN RETURN 2
IF (ez < 17) THEN RETURN 1
IF (ez < 23) THEN RETURN 2
IF (ez = 23) THEN RETURN 0x10
IF (ez = 24) THEN RETURN 0x20
IF (ez = 25) THEN RETURN 0x30
IF (ez = 26) THEN RETURN 0x40
IF (ez = 27) THEN RETURN 0x50
IF (ez = 28) THEN RETURN 0x60
IF (ez = 29) THEN RETURN 0x70
IF (ez = 30) THEN RETURN 0x80
IF (ez = 31) THEN RETURN 0x90
PRINT "p1"
RETURN 0
```

```
FUNCTION EZValueToPart2(ez)
IF (ez = 0) THEN RETURN 0
IF (ez < 9) THEN RETURN ((ez-1)*16+0x80)
IF (ez < 17) THEN RETURN ((ez-9)*16)
IF (ez < 23) THEN RETURN ((ez-17)*16)
IF (ez < 32) THEN RETURN 0
PRINT "p2"
RETURN 0
```

#### 40.4.5 Raw Bluetooth commands

Writes a single red dot into position (2,2) on a Dotti device using the raw Bluetooth commands

```
CLS BLUE
PRINT "Write red dot onto DOTI device"
```

```
device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    device = devices.Get(i)
    IF (device.Name = "Dotti") THEN WriteDot(device, 10, 255, 0, 0)
END IF

FUNCTION WriteDot(bt, pos, r, g, b)
    bt.Init()
    REM The fff0 is the service for many DOTTI commands
    REM The fff3 is the characteristic used by service fff0
    REM      for many of the DOTTI commands
    REM the 7 and 2 are the bytes that define the DOTTI
    REM command to send (0x0702 means set LED color)
    REM the pos is the position from 1 to 64
    REM the r g and b are the color to set.
    bt.WriteBytes ("fff0", "fff3", 7, 2, pos, r, g, b)
RETURN
```

#### 40.4.6 SetAnimationSpeed

Sets the animation speed (and does a ChangeMode to the animation)

```
CLS BLUE
PRINT "Set Dotti animation speed"

device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    device = devices.Get(i)
    Dotti = device.As ("DOTTI")
    speed = INPUT DEFAULT 1 PROMPT "What speed (1 to 6)?"
    Status = Dotti.ChangeMode(1)
    Status = Dotti.SetAnimationSpeed(speed)
```

```
PRINT "status", Status  
END IF
```

#### 40.4.7 SetColumn and SetRow to random lines

Draw random color lines on a Dotti device using SetColumn and SetRow

```
CLS BLUE  
PRINT "Write random lines"  
  
device = Bluetooth.PickDevicesName("*Dotti")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    Dotti = device.As ("DOTTI")  
  
    CLS GREEN  
    PRINT device.Name  
    Dotti.SetPanel (50, 50, 50)  
    REM set to a medium kind of green  
    FOR n = 1 TO 200  
        x = Math.Ceiling(RND * 8)  
        green = Math.Ceiling(RND*255)  
        red = Math.Ceiling(RND*255)  
        blue = Math.Ceiling(RND*255)  
        Dotti.SetColumn (x, red, green, blue)  
  
        x = Math.Ceiling(RND * 8)  
        green = Math.Ceiling(RND*255)  
        red = Math.Ceiling(RND*255)  
        blue = Math.Ceiling(RND*255)  
        Dotti.SetRow (x, red, green, blue)  
  
    NEXT n  
END IF
```

## 40.4.8 SetName of a Dotti device

Sets the name of a Dotti device

```
CLS BLUE
PRINT "Change the name of a Dotti device"

device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    PRINT device.Name
    Dotti = device.As ("DOTTI")
    Status = Dotti.SetName ("Dotti")
    PRINT "status", Status
END IF
```

## 40.4.9 SetPixel to random green dots

Displays random green dots on the Dotti using the dotti.SetPixel command.

```
CLS BLUE
PRINT "Write green dot onto DOTI device"

device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Dotti = device.As ("DOTTI")

    CLS GREEN
    PRINT device.Name
    Dotti.SetPanel (0, 10, 0)
    REM set to a medium kind of green
    FOR n = 1 TO 200
        x = Math.Ceiling(RND * 8)
```

```
y = Math.Ceiling(RND*8)
green = Math.Ceiling(RND*255)
Status = Dotti.SetPixel (x, y, 0, green, 0)

NEXT n
END IF
```

#### 40.4.10 SetPixel to write a single green dot

Writes a green dot to a Dotti device using the dotti.SetPixel() command

```
CLS BLUE
PRINT "Write green dot onto DOTTI device"

device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Dotti = device.As ("DOTTI")
    Status = Dotti.SetPixel (3, 3, 0, 255, 0)
    PRINT "status", Status
END IF
```

#### 40.4.11 Sync Time

Sets the time on the Dotti device

```
CLS BLUE
PRINT "SyncTime -- set Dotti time"

device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Dotti = device.As ("DOTTI")
    PRINT "DOTTI", Dotti
    h = INPUT DEFAULT 10 PROMPT "What hour?"
    m = INPUT DEFAULT 10 PROMPT "What minute?"
```

```
s = INPUT DEFAULT 15 PROMPT "What second?"  
Status = Dotti.ChangeMode(2)  
Status = Dotti.SyncTime(h, m, s)  
PRINT "status", Status  
END IF
```

## 40.5 BT: HEXIWEAR

The Hexiwear is a small hexagonal "wearable" IOT device from <http://www.hexiwear.com/>. It includes a number of sensors including heart rate, steps, weather and the normal accelerometer and gyroscope. The Hexiwear device. As("Hexiwear") specialization gives you easy access to all of the Hexiwear data.

### 40.5.1 Accelerometer

This program provides a constant stream of accelerometer updates from the device. In the program, all of the Hexiwear devices are listed (a device is known to be a Hexiwear device if it's name is HEXIWEAR). For each device found, the device. As("Hexiwear") specialization is created. Then the program goes into a loop, getting the data and printing it to the screen.

```
CLS BLUE  
PRINT AT 5,1 "Demonstrate Hexiwear Accelerometer"  
  
device = Bluetooth.PickDevicesName("HEXIWEAR")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    tag = device.As("Hexiwear")  
    PRINT AT 6,1 "Got a device", device.Name  
  
  
REM Now poll for data.  
FOR time = 1 TO 10  
    Screen.ClearLine (3)  
    PRINT AT 3, 1 "TIME", time
```

```
Screen.ClearLine(1)
data = tag.GetAccelerometer()
PRINT AT 1,1 data.Get(1), data.Get(2), data.Get(3)

PAUSE 50
NEXT time

PRINT AT 9, 1 "FINISH", status
tag.Close()
END IF
```

## 40.5.2 Compass

This program provides a constant stream of compass updates from the device. In the program, all of the Hexiwear devices are listed (a device is known to be a Hexiwear device if it's name is HEXIWEAR). For each device found, the device.As("Hexiwear") specialization is created. Then the program goes into a loop, getting the data and printing it to the screen.

```
CLS BLUE
PRINT AT 7,1 "Demonstrate Hexiwear Magnetometer"

device = Bluetooth.PickDevicesName("HEXIWEAR")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("Hexiwear")
    PRINT AT 8,1 "Got a device", device.Name

    data = tag.GetMagnetometer()
    REM Now poll for data.
    FOR time = 1 TO 99
        Screen.ClearLine(11)
        PRINT AT 11, 1 "TIME", time
        data = device.ReadRawBytes("2000", "2003")
```

```
FOR i=1 TO 6
    Screen.ClearLine(1)
    PRINT AT i,1 data.Get(i)
NEXT i

GOTO 90
data = tag.GetMagnetometer()
PRINT AT 1,1 INT (data.Heading)
Screen.ClearLine(3)
PRINT AT 3,1 data.X, data.Y
Screen.ClearLine(4)
PRINT AT 4,1 data.Z
90 REM bottom
PAUSE 10
NEXT time

PRINT AT 11, 1 "FINISH", status
tag.Close()
END IF
```

#### 40.5.3 List Information

The List Information program provides information about each Hexiwear device. For each device, a device.As("Hexiwear") specialization is created. The program then prints the device name, battery power level, manufacturer name and firmware revision. To see all of kinds of data you can read from a Hexiwear device, look at the ReadAll program. It prints all of the data that a Hexiwear is capable of producing.

```
CLS BLUE
PRINT "Available Bluetooth devices"

devices = Bluetooth.DevicesName ("HEXIWEAR")

FOR i = 1 TO devices.Count
```

```
device = devices.Get(i)
PRINT "NAME", device.Name
tag = device.As("Hexiwear")
PRINT tag.GetName()
PRINT tag.GetPower()
PRINT tag.GetManufacturerName()
REM does not work. PRINT tag.GetHardwareRevision()
PRINT tag.GetFirmwareRevision()
```

NEXT i

```
n = devices.Count
PRINT " "
PRINT "" + n + " devices were found"
```

#### 40.5.4 Raw Access to Hexiwear

The Raw Access program demonstrates how you can get information from a Hexiwear device without using the specialization. To do this, you will need know the different Bluetooth services and characteristics that a Hexiwear device exposes and how to read the resulting data. This documentation is available at

<https://www.dropbox.com/s/92tphuymsv0n5kx/HEXIWEAR%20Bluetooth%20Specifications.pdf?dl=0> In this program the accelerometer data is read. The acceleration service is server "2000" and the acceleration data is characteristic "2001". The data is read using the ReadRawBytes() method; that method returns an array of 6 bytes. The array starts at index 1. The bytes of the array must be interpreted as 3 16-bit integers. To interpret the bytes, you can use the built-in GetValue() method on the data array; that method takes in two parameters. The first parameter is the index to start reading at and the second is the interpretation type. Use "int16-le" to interpret the data as a 16-bit signed integer, little endian.

```
CLS BLUE
ACCSERVICE ="2000"
ACCDATA ="2001"
```

```
device = Bluetooth.PickDevicesName("HEXIWEAR")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    PRINT "DEVICE", device.Name

    address= device.Init()
    PRINT "Address", address

    REM Don't have to tell the device to turn on accelerometer

    PRINT "X", "Y", "Z"
    FOR time = 1 TO 15
        PAUSE 50
        data = device.ReadRawBytes(ACCSERVICE, ACCDATA)
        x = data.GetValue (1, "int16-le") / 100
        y = data.GetValue(3, "int16-le") / 100
        z = data.GetValue(5, "int16-le") / 100
        PRINT x, y, z
    NEXT time

    PRINT "Done"
END IF
```

#### 40.5.5 Read All

The Read All program demonstrates all of the different sensors in the Hexiwear IOT device. In the program, all of the Hexiwear devices are listed and a specialization created. Then the Hexiwear mode is read. Depending on the mode, the heart rate, pedometer or sensor data will be printed.

```
CLS BLUE
PRINT AT 12,1 "Demonstrate Hexiwear sensors"
```

```
device = Bluetooth.PickDevicesName("HEXIWEAR")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("Hexiwear")
    PRINT AT 1,1 "Got a device", device.Name

REM Now poll for data.
FOR time = 1 TO 4
    Screen.ClearLine(2)
    PRINT AT 2, 1 "TIME", time

    mode = tag.GetMode()
    PRINT AT 3,1 "MODE", mode

    IF (mode = 2) THEN ShowSensors(tag)
    IF (mode = 5) THEN ShowHeart(tag)
    IF (mode = 6) THEN ShowPedometer(tag)

    PAUSE 50
NEXT time

PRINT AT 11, 1 "FINISH", status
tag.Close()
END IF

FUNCTION ShowHeart(tag)
    Screen.ClearLine(4)
    PRINT AT 4,1 "Heart", tag.GetHeart()
RETURN

FUNCTION ShowPedometer(tag)
    Screen.ClearLine(4)
    Screen.ClearLine(5)
    PRINT AT 4,1 "Steps", tag.GetSteps()
    PRINT AT 5,1 "Calorie", tag.GetCalories()
```

**RETURN**

```
FUNCTION ShowSensors(tag)
    value = tag.GetAccelerometer()
    Screen.ClearLine(4)
    PRINT AT 4,1 "Accel.", "" + value.X + " " + value.Y + " " + value.Z

    value = tag.GetGyroscope()
    Screen.ClearLine(5)
    PRINT AT 5,1 "Gyro.", "" + value.X + " " + value.Y + " " + value.Z

    value = tag.GetMagnetometer()
    Screen.ClearLine(6)
    PRINT AT 6,1 "Mag.", "" + value.X + " " + value.Y + " " + value.Z

    Screen.ClearLine(7)
    PRINT AT 7,1 "Temp", tag.GetTemperature()
    Screen.ClearLine(8)
    PRINT AT 8,1 "Humidity", tag.GetHumidity()
    Screen.ClearLine(9)
    PRINT AT 9,1 "Pressure", tag.GetPressure()
    Screen.ClearLine(10)
    PRINT AT 10,1 "Light", tag.GetLight()
```

**RETURN**

#### 40.5.6 Set notification count

A new program for you to edit

```
CLS BLUE
PRINT "Demonstrate Hexiwear SetNotificationCount"

device = Bluetooth.PickDevicesName("HEXIWEAR")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
```

```
tag = device.As("Hexiwear")
PRINT "Got a device", device.Name

REM Now set the notification
REM 2=missed call 4=social 6=email
REM second value is the count to set.
status = tag.SetNotificationCount(6, 17)
PRINT ". status", status
END IF
```

## 40.5.7 SetTime

A new program for you to edit

```
CLS BLUE
PRINT "Demonstrate Hexiwear SetTimeNow"

device = Bluetooth.PickDevicesName("HEXIWEAR")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("Hexiwear")
    PRINT "Got a device", device.Name

    REM Now set the time
    status = tag.SetTimeNow()
    PRINT ". status", status
END IF
```

## 40.6 BT: MAGICLIGHT

Supports the MagicLight and Flux lights. These are Bluetooth-enabled light bulbs for home use.

### 40.6.1 Green

Turns the device green

```
CLS BLUE
```

```
PRINT "Sets the light to green"
```

```
device = Bluetooth.PickDevicesName("LEDBlue*")
```

```
IF (device.IsError)
```

```
    PRINT "No device was picked"
```

```
ELSE
```

```
    light = device.As ("MagicLight")
```

```
REM The three parameters are Red, Green and Blue values.
```

```
REM They must be in the range 0 to 255
```

```
Status = light.SetColor (0, 255, 0)
```

```
PRINT "status", Status
```

```
END IF
```

## 40.6.2 Off

Turns the device off

```
CLS BLUE
```

```
PRINT "Turns the light off"
```

```
device = Bluetooth.PickDevicesName("LEDBlue*")
```

```
IF (device.IsError)
```

```
    PRINT "No device was picked"
```

```
ELSE
```

```
    light = device.As ("MagicLight")
```

```
Status = light.SetOff ()
```

```
PRINT "status", Status
```

```
END IF
```

## 40.6.3 On

Turns the device on

```
CLS BLUE
```

```
PRINT "Turns the light on"
```

```
device = Bluetooth.PickDevicesName("LEDBlue*")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    light = device.As ("MagicLight")  
    Status = light.SetOn ()  
    PRINT "status", Status  
END IF
```

#### 40.6.4 Red

Turns the light red

```
CLS BLUE  
PRINT "Sets the light to red"  
  
device = Bluetooth.PickDevicesName("LEDBlue*")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    light = device.As ("MagicLight")  
    REM The three parameters are Red, Green and Blue values.  
    REM They must be in the range 0 to 255  
    Status = lightSetColor (255, 0, 0)  
    PRINT "status", Status  
END IF
```

### 40.7 BT: SENZORTAG 1350

Demonstrates how to use the TI SensorTag 1350 (a V2 version released in late 2016). The model 1350 SensorTag from Texas Instruments is a small, battery-powered sensor platform from TI. The sensors include an accelerometer, gyroscope, IR contactless thermometer, humidity sensor, magnetometer, barometer and on-chip temperature sensor. It also includes a light sensor and a magnetic switch detector (reed relay).

#### 40.7.1 Accelerometer Gyroscope and Magnetometer

Demonstrates the basics of the AccelerometerSetup and using a callback routine. The V2 SensorTag has a combined accelerometer/gyroscope/magnetometer chip that provides XYZ data for all three sensors at once.

```
CLS BLUE
```

```
PRINT AT 5,1 "Demonstrate TI SensorTag Accelerometer"
```

```
device = Bluetooth.PickDevicesName("CC1350 SensorTag,SensorTag  
2.0")
```

```
IF (device.IsError)
```

```
    PRINT "No device was picked"
```

```
ELSE
```

```
    tag = device.As("SensorTag1350")
```

```
    PRINT AT 6,1 "Got a device", device.Name
```

```
    REM TABLE: Bits to turn on different position sensors
```

```
    REM 1 = Gyro Z axis
```

```
    REM 2 = Gyro Y axis
```

```
    REM 4 = Gyro X axis [7==Gyro ALL axis]
```

```
    REM 8 = Acc X axis
```

```
    REM 16 = Acc Y axis
```

```
    REM 32 = Acc Z axis [56==Acc ALL axis]
```

```
    REM 64 = Mag ALL axis
```

```
    REM 128 = Wake-on-motion enabled
```

```
    REM 0 = 2G range on acc
```

```
    REM 256 = 4G range on acc
```

```
    REM 512 = 8G range on acc
```

```
    REM 768 = 16G range on acc
```

```
    REM Example: to turn on all axis of the acc and nothing else with a  
4G range:
```

```
    REM AccFlag = 8+16+32+256
```

```
    REM Turn on all devices, no wake-on-movement, acc range 2G.
```

```
    AccFlag = 1+2+4+8+16+32+64
```

```
AccFlag = 8+16+32
```

```
PRINT AT 7,1 "SETUP", tag.AccelerometerSetup(AccFlag, 20, "Acc")
```

```
PRINT AT 8,1 "Done with setup"
```

```
REM Now wait a little while. The Acc routine will  
REM be called with updates.
```

```
FOR time = 1 TO 10
```

```
    Screen.ClearLine(1)
```

```
    now = DateTime.GetNow()
```

```
    PRINT "TIME", now.Time
```

```
    PAUSE 50
```

```
NEXT time
```

```
REM Undo the accelerometer
```

```
status = tag.AccelerometerSetup(0, 0, "")
```

```
PRINT AT 9, 1 "FINISH", status
```

```
tag.Close()
```

```
END IF
```

```
FUNCTION Acc(tag, ax, ay, az, mx, my, mz, rx, ry, rz)
```

```
    Screen.ClearLine(2)
```

```
    PRINT "X", "Y", "Z"
```

```
    Screen.ClearLine(3)
```

```
    PRINT Math.Round(ax,2), Math.Round(ay,2), Math.Round(az,2)
```

```
    Screen.ClearLine(4)
```

```
    PRINT Math.Round(mx,2), Math.Round(my,2), Math.Round(mz,2)
```

```
    Screen.ClearLine(5)
```

```
    PRINT Math.Round(rx,2), Math.Round(ry,2), Math.Round(rz,2)
```

```
RETURN
```

## 40.7.2 Accelerometer Off

Turns off the accelerometer

**CLS BLUE**

```
device = Bluetooth.PickDevicesName("CC1350 SensorTag,SensorTag  
2.0")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    tag = device.As("SensorTag1350")  
    PRINT "Got a device", device.Name  
    tag.SetupAcc(0, 100, "Acc")  
END IF  
PRINT "All done"
```

#### 40.7.3 Barometer

Demonstrates the basics of the BarometerSetup and using a callback routine.

**CLS BLUE**

```
PRINT AT 1,1 "Demonstrate SensorTag Barometer measurements"
```

```
REM device = Bluetooth.PickDevicesName("CC1350  
SensorTag,SensorTag 2.0")  
devices = Bluetooth.DevicesName("CC1350 SensorTag")  
device = devices[1]  
  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    tag = device.As("SensorTag1350")  
    REM 100=1000ms=1 second  
    PRINT AT 7,1 "SETUP", tag.BarometerSetup(1, 100, "Barometer")  
  
    FOR time = 1 TO 30  
        PAUSE 50  
        now = DateTime.GetNow()  
        Screen.ClearLine(2)
```

```
PRINT "TIME", now.Time
NEXT time

PRINT AT 8,1 "CLOSE", tag.BarometerSetup(0, 100, "Barometer")
END IF

REM Temperatures are in degrees C
REM pressure is in hpa
FUNCTION Barometer(tag, temp, pressure)
    Screen.ClearLine(3)
    PRINT "Temp", temp, CTOF(temp)
    Screen.ClearLine(4)
    PRINT "Pressure", pressure, HPATOINCHM(pressure)
RETURN

FUNCTION CTOF(C)
    F = C * 9/5 + 32
    F = Math.Round(F, 1)
RETURN F

FUNCTION HPATOINCHM(HPA)
    ATM = HPA / 1013.25
    INCHM = ATM * 29.9213
    INCHM = Math.Round(INCHM, 2)
RETURN INCHM
```

#### 40.7.4 Button

Demonstrates the "Simple Key Service" on the SensorTag

```
CLS BLUE
PRINT AT 5,1 "Demonstrate SensorTag Buttons"
PRINT AT 6,1 "Count", devices.Count

device = Bluetooth.PickDevicesName("CC1350 SensorTag,SensorTag
2.0")
IF (device.IsError)
```

```
PRINT "No device was picked"  
ELSE  
    tag = device.As("SensorTag1350")  
    PRINT AT 7,1 "SETUP", tag.ButtonSetup(1, "Button")  
  
    FOR time = 1 TO 30  
        PAUSE 50  
        PRINT AT 3,1 "TIME", time  
    NEXT time  
  
    PRINT AT 8,1 "CLOSE", tag.ButtonSetup(0, "Button")  
END IF  
  
FUNCTION Button(tag, left, right, side)  
    Screen.ClearLine(1)  
    IF (left) THEN PRINT AT 1,1 "LEFT"  
    IF (right) THEN PRINT AT 1,8 "RIGHT"  
    IF (side) THEN PRINT AT 1,16 "SIDE"  
RETURN
```

## 40.7.5 Humidity

Demonstrates the basics of the HumiditySetup and using a callback routine.

```
CLS BLUE  
PRINT AT 1,1 "Demonstrate SensorTag Humidity measurements"  
  
FOR i=1 TO devices.Count  
    device = devices.Get(i)  
    tag = device.As("SensorTag1350")  
    REM 100=1000ms=1 second  
    PRINT AT 7,1 "SETUP", tag.HumiditySetup(1, 100, "Humidity")  
  
    FOR time = 1 TO 30  
        PAUSE 50  
        now = DateTime.GetNow()
```

```
Screen.ClearLine(1)
PRINT "TIME", now.Time
NEXT time

PRINT AT 8,1 "CLOSE", tag.HumiditySetup(0, 100, "Humidity")
NEXT i

REM Temperatures are in degrees C
FUNCTION Humidity(tag, temp, humidity)
    Screen.ClearLine(2)
    PRINT "TEMP", Math.Round(temp,1), CTOF(temp)

    Screen.ClearLine(3)
    PRINT "Humidity", Math.Round(humidity,2)
RETURN

FUNCTION CTOF(C)
    F = C * 9/5 + 32
    F = Math.Round(F, 1)
RETURN F
```

## 40.7.6 IO

Lets you control the devices on the SensorTag. The 1350 includes a red LED and a buzzer; the 2650 includes both a red and a green LED.

```
CLS BLUE
PRINT AT 1,1 "Demonstrate SensorTag LED/Buzzer control"

device = Bluetooth.PickDeviceName("CC1350 SensorTag,SensorTag
2.0")

IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("SensorTag1350")
    val = INPUT DEFAULT 1 PROMPT "1=RED 4=BUZZER 0=Both off"
```

```
    tag.IO(val)  
END IF
```

## 40.7.7 IR

Demonstrates the basics of the IRSetup and using a callback routine.

```
CLS BLUE  
PRINT AT 1,1 "Demonstrate SensorTag IR measurements"  
  
device = Bluetooth.PickDevicesName("CC1350 SensorTag,SensorTag  
2.0")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    tag = device.As("SensorTag1350")  
    REM 100=1000ms=1 second  
    PRINT AT 7,1 "SETUP", tag.IRSetup(1, 100, "IR")  
  
    FOR time = 1 TO 30  
        PAUSE 50  
        now = DateTime.GetNow()  
        Screen.ClearLine(2)  
        PRINT "TIME", now.Time  
    NEXT time  
  
    PRINT AT 8,1 "CLOSE", tag.IRSetup(0, 100, "IR")  
END IF  
  
REM Temperatures are in degrees C  
FUNCTION IR(tag, objTemp, ambTemp)  
    Screen.ClearLine(3)  
    PRINT "Object", Math.Round(objTemp,1), CTOF(objTemp)  
  
    Screen.ClearLine(4)  
    PRINT "Ambient", Math.Round(ambTemp,1), CTOF(ambTemp)  
RETURN
```

```
FUNCTION CTOF(C)
  F = C * 9/5 + 32
  F = Math.Round(F*10) / 10
RETURN F
```

## 40.7.8 Optical Sensor

A new program for you to edit

```
CLS BLUE
PRINT AT 1,1 "Demonstrate SensorTag Optical measurements"

device = Bluetooth.PickDevicesName("CC1350 SensorTag,SensorTag
2.0")

IF (device.IsError)
  PRINT "No device was picked"
ELSE
  tag = device.As("SensorTag1350")
  REM 100=1000ms=1 second
  PRINT AT 7,1 "SETUP", tag.OpticalSetup(1, 100, "Optical")

  FOR time = 1 TO 30
    PAUSE 50
    now = DateTime.GetNow()
    Screen.ClearLine(2)
    PRINT "TIME", now.Time
  NEXT time

  PRINT AT 8,1 "CLOSE", tag.OpticalSetup(0, 100, "Optical")
END IF

REM Light sensor readings are in Lux
FUNCTION Optical(tag, lux)
  Screen.ClearLine(3)
  PRINT "LUX", lux
```

**RETURN**

## 40.8 EX: BC BASIC QUICK SAMPLES

A set of the most common programs people need. Includes a tip program, money conversion, miles per gallon, and more.

### 40.8.1 Colorful Countdown

A bright countdown display. Will count down for the number of seconds in the calculator window. The minimum countdown is 5 seconds, and the maximum is 60 seconds.

```
value = Calculator.Value
IF value < 5 THEN value = 5
IF value > 60 THEN value = 60

PRINT "Count down!"

REM anitime sets the speed of the color changes
REM when set low (like 5), the colors really flash quickly
REM when set to 50, the color changes with the display
anitime = 25

ctime = 0
display = value
FOR i=0 TO (value*50) STEP anitime
color = color + 1
IF color >= 7 THEN color = 1
CLS color
PrintTitle (1, "Colorful Countdown")
PrintCenter (display)
PAUSE anitime
ctime = ctime + anitime
IF ctime < 50 THEN NEXT i
ctime = ctime - 50
display = display - 1
```

**NEXT i****PrintCenter ("Countdown Complete")**

```
FUNCTION PrintTitle(row, str)
Imargin = 1+INT (( Screen.W - LEN str) / 2)
IF (Imargin < 1) THEN Imargin = 1
PRINT AT row,Imargin str
RETURN
```

```
FUNCTION PrintCenter (str)
Imargin = 1+INT (( Screen.W - LEN str) / 2)
IF (Imargin < 1) THEN Imargin = 1
row = INT ((Screen.H) / 2)
PRINT AT row,Imargin str
RETURN
```

## 40.8.2 Grams of Fat to Calories

Takes the value already in the calculator and converts it from grams of fat to calories. This program assumes that all fat is 9 calories per gram.

```
value = Calculator.Value
retval=value * 9
Calculator.Message = "Converted " + value + " grams of fat to
calories"

CLS BLUE
PRINT "Convert Grams of Fat to Calories"
PRINT "Input="; value
PRINT " "
PRINT "There are 9 calorie per gram of fat"
PRINT " "
PRINT "Calories="; retval
STOP retval
```

## 40.8.3 Miles per Gallon

Calculates mile per gallon given number of miles driven and total gallons of gas.

```
CLS
PRINT "Calculating Miles Per Gallon"
PRINT " "
miles = INPUT DEFAULT 100 PROMPT "How many miles were driven?"
gallons = INPUT DEFAULT 4 PROMPT "How many gallons did you
need?"
retval = mpg(miles, gallons)

PRINT "Miles driven="; miles
PRINT "Gallons used="; gallons
PRINT "MPG="; retval

IF NOT Memory.IsSet ("PreviousMpg") THEN GOTO 40
lastMpg = Memory.PreviousMpg
deltaMpg = retval - lastMpg
PRINT "Last time="; lastMpg
IF ABS (deltaMpg) > 1.5 THEN GOTO 10
PAPER GREEN
PRINT "MPG is about the same"
GOTO 40

10 IF deltaMpg >= 1.5 THEN GOTO 20

PAPER RED
PRINT "MPG has decreased!"
GOTO 40

20 PAPER GREEN
PRINT "MPG has increased!"
GOTO 40

40 REM
```

```
Memory.PreviousMpg = retval  
STOP retval
```

```
FUNCTION mpg (Miles, Gallons)  
Retval = Miles / Gallons  
RETURN Retval
```

#### 40.8.4 Right Triangle calculator

Uses the Pythagorean theorem to calculate the hypotenuse of a right triangle based on the other two sides.

```
REM Calculate the hypotenuse of a triangle  
CLS BLUE  
PRINT "Right triangle calculator"  
PRINT ""  
  
A = INPUT DEFAULT 3 PROMPT "Enter the first side"  
B = INPUT DEFAULT 4 PROMPT "Enter the second side"  
C = hypotenuse (A, B)  
  
PRINT "First side=";A  
PRINT "Second side=";B  
PRINT "Hypotenuse=";C  
PRINT ""  
PRINT "Calculation is √(A**2 + B**2)"  
STOP C  
  
REM Calculate the hypotenuse from A and B  
FUNCTION hypotenuse (A, B)  
C=2 √(A**2 + B**2)  
RETURN C
```

#### 40.8.5 Tip Calculator

A new program for you to edit

```
value = Calculator.Value
```

```
CLS GREEN
```

```
TEST()
```

```
PRINT "Tip Calculator"
```

```
PRINT " "
```

```
PRINT " 5% tip of "; value; " is "; Tip(value, 5)
```

```
PRINT " 10% tip of "; value; " is "; Tip(value, 10)
```

```
PRINT " 15% tip of "; value; " is "; Tip(value, 15)
```

```
PRINT " 18% tip of "; value; " is "; Tip(value, 18)
```

```
PRINT " 20% tip of "; value; " is "; Tip(value, 20)
```

```
PRINT " "
```

```
Calculator.Message = "15% tip of " + value + " is " + Tip(value, 15)
```

```
STOP 0+Tip(value, 15)
```

```
REM We need a fancy function because we need to format the number
```

```
REM nice and neat. It should be calculated to the nearest penny
```

```
exactly.
```

```
FUNCTION Tip(value, percent)
```

```
raw = value * (percent/100)
```

```
round = Math.Round (raw * 100) / 100
```

```
fraction = round - Math.Truncate(round)
```

```
fraction = Math.Round (fraction * 100)
```

```
IF (fraction < 10) THEN fraction = "0" + fraction
```

```
top = "" + Math.Truncate(round) + "." + fraction
```

```
RETURN top
```

```
FUNCTION TestOne (value, percent, expected)
```

```
actual = Tip (value, percent)
```

```
IF (actual ≈ expected) THEN RETURN 0
```

```
PRINT "ERROR; TIP ";value; " pct "; percent
```

```
PRINT "Expected "; expected
```

```
PRINT "Actual "; actual
```

```
PRINT "Difference "; actual-expected
```

```
RETURN 1
```

```
FUNCTION TEST ()
```

```
nerror = 0  
REM Tip returns a string, not a number  
nerror = nerror + TestOne (100, 5, "5.00")  
nerror = nerror + TestOne (76, 15, "11.40")  
nerror = nerror + TestOne (140, 15, "21.00")
```

```
IF (nerror > 0) THEN PRINT "HORIZON NERROR=";nerror  
IF (nerror > 0) THEN PAPER RED  
RETURN nerror
```

## 40.8.6 Welcome to BC BASIC

Describes BC BASIC for new users

```
CLS BLUE  
PRINT "WELCOME TO BC BASIC!"  
PRINT " "  
PRINT "You can program the P1 to P5 keys"  
PRINT "to perform ANY function you want"  
PRINT "using Best Calculator BASIC"  
PRINT " "  
PRINT "Tap the BC BASIC button to get started"  
PRINT " - there is full help available"  
PRINT " - there are lots of samples"  
PRINT " - you can get started right away"
```

## 40.9 EX: FILES, CSV AND JSON, HTML, FLOW

Demonstrates how to read and write files, including CSV and JSON data using the File object and the String.Escape and String.Parse function.

Demonstrates how to use the HTML functionality and includes a longer example with Microsoft Flow.

### 40.9.1 Appending to a file

Demonstrates picking and appending to a file. Each call to AppendLine() and AppendText() will write to the end of the file.

REM

REM Demonstrate AppendPicker, AppendText and AppendLine

REM

```
file = File.AppendPicker("CSV file", ".csv", "test.csv")
```

IF (file.IsError)

    REM file will have a error message

    PRINT file

    STOP

END IF

PRINT "SIZE", file.Size()

IF (file.Size( )= 0) THEN file.AppendLine("time,data")

now = DateTime.GetNow()

REM

REM Use an array to make

REM perfect CSV data

REM

DIM data(2)

data(1) = now.Time

data(2) = 42.42

file.AppendText (String.Escape("csv", data))

#### 40.9.2 Http.Get(url, headers) reads data from the internet

Demonstrates downloading data from the internet using Http.Get(url, headers). The resulting JSON data is parsed into an array.

REM Demonstrate downloading from the internet

REM

REM Download content from a news feed

REM Make sure the download was OK

REM Parse the JSON into data

REM

```
url = "https://hacker-
```

```
news.firebaseio.com/v0/item/8863.json?print=pretty"
```

```
result = Http.Get(url)
IF (result.IsError)
    REM Did not get data
    CLS RED
    PRINT "Unable to download URL"
    PRINT "ErrorCode", result.ErrorCode
    PRINT "ErrorString", result.ErrorString

ELSE
    REM All OK
    CLS GREEN
    PRINT "Downloaded from URL"
    PRINT "Status", result.StatusCode
    PRINT "Reason", result.ReasonPhrase
    REM PRINT "Content", result.Content

    REM Now parse it as json
    REM You can pull individual bits out
    data = String.Parse("json", result.Content)
    PRINT "data.by", data.by
    PRINT "data.title", data.title
    PRINT "data", data.Count

    REM You can also pull data by index
    FOR i=1 TO data.Count
        PRINT i, data[i]
    NEXT i

END IF
```

### 40.9.3 Microsoft Flow example

A longer example showing how to trigger Microsoft Flow using HTML. Data is put into an array and converted to JSON format using `String.Escape ("json", list)`; the value is then sent to a Microsoft trigger HTML endpoint using the `Http.Post(url, data, headers)` method.

CLS BLUE

REM

REM The Microsoft Flow trigger URL is stored in the memory area

REM

memory = "Microsoft.Flow Example URL"

url = Memory.GetOrDefault (memory, "")

url = INPUT DEFAULT url PROMPT "Microsoft Flow URL"

Memory[memory] = url

REM

REM Set up the constant monitoring values

REM

min = 30

max = 40

deviceName = "My device"

sensor = "temperature"

REM

REM Set up the sensor device.

REM This program uses data from the MetaWear device

REM

device = Bluetooth.PickDevicesName ("MetaWear")

IF device.IsError

    CLS RED

    PRINT "No device picked"

    STOP

END IF

meta = device.As ("MetaMotion")

meta.TemperatureSetup(1, "Temperature")

REM

REM Main loop; will keep on spinning and

REM asking for updated temperature readings.

REM

ExitRequested = 0

```
MAXTIME=1000
FOR time=0 TO MAXTIME
    PAUSE 50
    meta.TemperatureRead()
    IF (ExitRequested > 0) THEN time = MAXTIME
NEXT time

REM
REM Callback when temperature changes
REM
FUNCTION Temperature(ble, celcius)
    GLOBAL url
    GLOBAL deviceName
    GLOBAL sensor
    GLOBAL min
    GLOBAL max

    time = DateTime.GetNow()
    REM Convert to Fahrenheit
    data = celcius * 9 / 5 + 32

    Screen.ClearLine (9)
    Screen.ClearLine (10)
    Screen.ClearLine (11)
    PRINT AT 9,2 "TIME", time.Time
    PRINT AT 10,2 "TEMP", data

    IF (data < min OR data > max)
        PRINT AT 11,1 "SENDING DATA"
        SendData (url, data, time, deviceName, sensor, min, max)
        GLOBAL ExitRequested
        ExitRequested = 1
    END IF
RETURN

REM
REM Format and send data to Microsoft Flow
```

REM

FUNCTION SendData(url, data, time, deviceName, sensor, min, max)

REM

REM Put the data into correct JSON form

REM

DIM datalist()

datalist.AddRow ("data", data)

datalist.AddRow ("time", time)

datalist.AddRow ("device", deviceName)

datalist.AddRow ("sensor", sensor)

datalist.AddRow ("min", min)

datalist.AddRow ("max", max)

json = String.Escape ("json", datalist)

PRINT json

REM Microsoft Flow demands that data be passed using the

REM a Content-Type of application/json.

DIM header()

header[1] = "Content-Type: application/json"

result = Http.Post (url, json, header)

RETURN result

#### 40.9.4 Read Entire File

Demonstrates how to use File.ReadPicker to pick and read an entire file

REM

REM Demonstrate the File.ReadPicker

REM

CLS GREEN

PRINT "Demonstrate reading a file"

file = File.ReadPicker (".txt")

IF (file.IsError)

REM file has an error message

PRINT "file.IsError is TRUE"

```
PRINT file
STOP
END IF
PRINT "Size is ", file.Size()

REM ReadAll will read the entire file as single text.
fulltext = file.ReadAll()
PRINT "The entire file"
PRINT fulltext
PRINT " "

REM
REM ReadLines will read the entire file and split it
REM into individual lines.
REM
lines = file.ReadLines()

PRINT "Count of lines", lines.Count
IF (lines.Count > 1) THEN PRINT "First line", lines[1]
```

#### 40.9.5 Reading a CSV file

Uses the File.ReadPicker() to pick a CSV file. It's read in and parsed using String.Parse ("csv", data-string) and the results are printed

```
CLS BLUE
file = File.ReadPicker (".csv")
IF (file.IsError)
    REM file will contain an error string
    PRINT "ERROR", file
    STOP
END IF

alltext = file.ReadAll()
REM will print several lines of data
PRINT "All text", alltext
```

```
csv = String.Parse ("csv", alltext)
header = csv[1]
data = csv[2]
PRINT "HEADER", header(1), header(2)
FOR index = 2 TO csv.Count
    data = csv(index)
    PRINT index-1, data(1), data(2)
NEXT index
```

#### 40.9.6 Writing to a file (including CSV)

Demonstrates picking and writing to a file. The first WriteText to a picked file will overwrite the contents; after that each additional WriteText will append to the file. It's easy to make a CSV (comma seperated file) using the String.Escape("csv", data) method.

```
REM
REM Demonstrate WritePicker, WriteText and WriteLine
REM

file = File.WritePicker("CSV file", ".csv", "test.csv")
IF (file.IsError)
    REM file will have a error message
    PRINT file
    STOP
END IF

file.WriteLine("time,data")
now = DateTime.GetNow()

REM
REM Use an array to make
REM perfect CSV data
REM
DIM data(2)
data(1) = now.Time
data(2) = 42.42
```

## 40.10 EX: FINANCIAL

Sample financial programs for ROI (Return on Investment), Present and Future value, and more.

### 40.10.1 Common Tip Values

Starts with the value already in your calculator, and comes up with a range of tips (5%, 10%, 15%, 18%, 20%)

```
value = Calculator.Value  
CLS BLACK  
TEST()  
PRINT "5% tip of "; value; " is "; Tip(value, 5)  
PRINT "10% tip of "; value; " is "; Tip(value, 10)  
PRINT "15% tip of "; value; " is "; Tip(value, 15)  
PRINT "18% tip of "; value; " is "; Tip(value, 18)  
PRINT "20% tip of "; value; " is "; Tip(value, 20)
```

```
Calculator.Message = "15% tip of " + value + " is " + Tip(value, 15)  
STOP 0+Tip(value, 15)
```

REM We need a fancy function because we need to format the number  
REM nice and neat. It should be calculated to the nearest penny  
exactly.

```
FUNCTION Tip(value, percent)  
raw = value * (percent/100)  
round = Math.Round (raw * 100) / 100  
fraction = round - Math.Truncate(round)  
fraction = Math.Round (fraction * 100)  
IF (fraction < 10) THEN fraction = "0" + fraction  
top = "" + Math.Truncate(round) + "." + fraction  
RETURN top
```

```
FUNCTION TestOne (value, percent, expected)
```

```
actual = Tip (value, percent)
IF (actual ≈ expected) THEN RETURN 0
PRINT "ERROR; TIP ";value; " pct "; percent
PRINT "Expected "; expected
PRINT "Actual "; actual
PRINT "Difference "; actual-expected
RETURN 1
```

```
FUNCTION TEST ()
nerror = 0
REM Tip returns a string, not a number
nerror = nerror + TestOne (100, 5, "5.00")
nerror = nerror + TestOne (76, 15, "11.40")
nerror = nerror + TestOne (140, 15, "21.00")

IF (nerror > 0) THEN PRINT "HORIZON NERROR=";nerror
IF (nerror > 0) THEN PAPER RED
RETURN nerror
```

## 40.10.2 Compound Interest

Calculates compound interest

```
REM
REM Calculates the interest earned on a loan.
REM Loan terms and interest is given per year; the
REM interest is compounded monthly.
REM
```

```
TEST()
P = INPUT DEFAULT 1000 PROMPT "Principal (original balance)"
RY = INPUT DEFAULT 12 PROMPT "Rate per year (enter 3% as 3)"
NY = INPUT DEFAULT 1 PROMPT "Number of years"
C = CompoundInterest(P, RY, NY)
Calculator.Message = "Compound interest earned"
STOP C
```

```
FUNCTION CompoundInterest(P, RY, NY)
```

```
R = RY / 1200
```

```
N = NY * 12
```

```
C = P * ((1 + R)**N - 1)
```

```
RETURN C
```

```
FUNCTION TestOne (P, RY, NY, expected)
```

```
actual = CompoundInterest(P, RY, NY)
```

```
actual = Math.Round (actual * 100) / 100
```

```
IF (actual ≈ expected) THEN RETURN 0
```

```
PRINT "ERROR; P ";P; " RY "; RY
```

```
PRINT "Expected "; expected
```

```
PRINT "Actual "; actual
```

```
PRINT "Difference "; actual-expected
```

```
RETURN 1
```

```
FUNCTION TEST ()
```

```
nerror = 0
```

```
nerror = nerror + TestOne (1200, 12.49, .5, 76.92)
```

```
nerror = nerror + TestOne (1100, 3.2, 2, 72.60)
```

```
IF (nerror > 0) THEN PRINT "CompoundInterest NERROR=";nerror
```

```
IF (nerror > 0) THEN PAPER RED
```

```
RETURN nerror
```

### 40.10.3 Doubling Time

Calculate the time it takes to double an investment given a rate of return. The rate of return is in percent; 12% is represented as 12.

```
REM Takes the value in the calculator as a percent (e.g., 12% is 12)
```

```
REM divides by 100 to get computer-type percents (0.12)
```

```
REM values are in YEARS, but interest is assumed to compound  
MONTHLY.
```

```
REM Example: at a rate of "6" (6%), money will double in about 11.58  
years.
```

```
CLS BLACK
```

```
TEST()
```

```
yr = Calculator.Value
```

```
dt = DoublingTime(yr)
```

```
Calculator.Message = "Doubling Time in years at " + yr + "% per year"
```

```
STOP dt
```

```
REM Doubling time in years given a per-year interest rate that  
compounds monthly
```

```
REM yr is percent interest rate; e.g., give 4.25% as 4.25
```

```
FUNCTION DoublingTime(yr)
```

```
r = yr / 100
```

```
REM divide by 12 to get the monthly rate
```

```
r = r / 12
```

```
dt = LN(2) / LN(1 + r)
```

```
REM dt starts off in months, but we want to present years, so divide  
by 12.
```

```
dt = dt / 12
```

```
Calculator.Message = "Doubling Time in years at " + yr + "% per year"
```

```
RETURN dt
```

```
FUNCTION TestOne (yr, expected)
```

```
actual = DoublingTime(yr)
```

```
actual = Math.Round (actual * 100) / 100
```

```
IF (actual ≈ expected) THEN RETURN 0
```

```
PRINT "ERROR; DoublingTime "; yr
```

```
PRINT "Expected "; expected
```

```
PRINT "Actual "; actual
```

```
PRINT "Difference "; actual-expected
```

```
RETURN 1
```

```
FUNCTION TEST ()
```

```
nerror = 0
```

```
nerror = nerror + TestOne (6, 11.58)
```

```
nerror = nerror + TestOne (6.35, 10.94)
```

```
IF (nerror > 0) THEN PRINT "DoublingTime NERROR=";nerror  
IF (nerror > 0) THEN PAPER RED  
RETURN nerror
```

#### 40.10.4 Future Value

Calculates the future value of money today given a period of time and an interest rate.

```
REM Future value uses the standard formula FV = PV * (1+r)**n  
REM where PV = present value (e.g., money to invest)  
REM r is the interest rate (3% is .03 in the formula, but this program  
lets the user enter '3' for 3%  
REM n is the number of periods. This must match the interest rate  
(e.g., either both are 'per year' or both are 'per month')
```

```
REM Good practice to run the TEST program to make sure the  
calculations are OK.
```

```
TEST()
```

```
PV = INPUT DEFAULT 900 PROMPT "Present value"  
n = INPUT DEFAULT 3 PROMPT "How far into the future (years)"  
r = INPUT DEFAULT 3 PROMPT "Interest rate (per year). Enter 3 for  
3%"  
r = r / 100  
FV = FutureValue(PV, n, r)  
Calculator.Message = "FV of " + PV + " at " + r*100 + "%"   
STOP FV
```

```
FUNCTION FutureValue(PV, n, r)  
FV = PV * ((1+r)**n)  
RETURN FV
```

```
FUNCTION TestOne(PV, n, r, expected)  
nerror = 0  
actual = FutureValue(PV, n, r)
```

```
IF actual ≈ expected THEN GOTO 10
nerror = nerror + 1
PRINT "ERROR: FutureValue"
PRINT "PV=";PV; " n="; n; " r=";r
PRINT "expected="; expected
PRINT "actual=";actual
10 REM
RETURN nerror

FUNCTION TEST()
nerror = 0
nerror = nerror + TestOne (1000, 5, .1, 1610.51)
RETURN nerror
```

## 40.10.5 Money Conversion

Simple program to convert from one currency to another. The program will always prompt for the conversion rate, but will remember the last conversion rate you used. This program does not go on-line to get the current set of conversion rates (it's not possible in BC BASIC)

```
REM
REM The defaults here are roughly the conversion rate from yen to
REM australian dollars. 1 yen is about 0.011 australian dollar;
REM 10000 yen is therefore about 110 australian dollars.
REM
rate = INPUT DEFAULT Memory.GetOrDefault ("ConversionRate",
0.011) PROMPT "Conversation rate <from> to <to> [e.g., yen to
australian dollars]"
Memory.ConversionRate = rate
amount = INPUT DEFAULT Memory.GetOrDefault
("ConversionAmount", 10000) PROMPT "Amount to convert [e.g.,
amount in yen]"
Memory.ConversionAmount = amount
value = amount * rate
Calculator.Message = "Convert " + amount + " at a rate of " + rate +
" is " + value
```

```
Calculator.Value = value
```

## 40.10.6 Present Value

Calculates the present value (PV) of a sum of money (the future value, FV) given an interest rate and the number of years in the future that the sum of money will be paid.

```
REM Future value uses the standard formula PV = FV / (1+r)**n  
REM where PV = present value (e.g., money to invest)  
REM FV is the value of the investment in the future  
REM r is the interest rate (3% is .03 in the formula, but this program  
lets the user enter '3' for 3%  
REM n is the number of periods. This must match the interest rate  
(e.g., either both are 'per year' or both are 'per month')
```

REM Good practice to run the TEST program to make sure the calculations are OK.

```
TEST()
```

```
FV = INPUT DEFAULT 900 PROMPT "Future value (amount of money in the future)"
```

```
n = INPUT DEFAULT 3 PROMPT "When will the money be paid"
```

```
r = INPUT DEFAULT 3 PROMPT "Interest rate (per year). Enter 3 for 3%"
```

```
REM Some people want to type 10 for 10%
```

```
r = r / 100
```

```
PV = PresentValue (FV, n, r)
```

```
Calculator.Message = "PV of " + FV + " at " + r*100 + "%"
```

```
STOP PV
```

```
FUNCTION PresentValue(FV, n, r)
```

```
PV = FV / ((1+r)**n)
```

```
RETURN PV
```

```
FUNCTION TestOne(FV, n, r, expected)
```

```
nerror = 0
```

```
actual = PresentValue(FV, n, r)
IF actual ≈ expected THEN GOTO 10
nerror = nerror + 1
PRINT "ERROR: PresentValue"
PRINT "FV=";FV; " n="; n; " r=";r
PRINT "expected="; expected
PRINT "actual=";actual
10 REM
RETURN nerror

FUNCTION TEST()
nerror = 0
nerror = nerror + TestOne (900, 3, .1, 676.18)
nerror = nerror + TestOne (570, 1, .1, 518.18)
nerror = nerror + TestOne (570, 3, .1, 428.25)
RETURN nerror
```

## 40.10.7 Return on Investment

Also called ROI, the return on investment shows the percentage return on an investment.

```
REM ROI
EARNINGS = INPUT DEFAULT 1100 PROMPT "Earnings on the
investment"
INITIAL = INPUT DEFAULT 1000 PROMPT "Initial investment"
ROI = (EARNINGS - INITIAL) / INITIAL
Calculator.Message = "ROI when earnings is " + EARNINGS + " on an
initial investment of " + INITIAL
STOP ROI
```

## 40.11 EX: GOPHER AND GOPHER-OF-THINGS

Demonstrates using BC BASIC as a Gopher Server. Examples range from the very simple to a full IOT (or Gopher-of-things) sample. By using the Gopher protocol, you can write programs on one computer and control

it using an ordinary Gopher clients. Gopher clients are available for most platforms including Windows, iOS and Android.

### 40.11.1 A simple GOPHER Program

Best Calculator can act as a Gopher server, serving up Gopher pages that you design in code. This simple example shows the most simple Gopher program with a few sub-pages that show the current system settings.

```
CLS
PRINT "Gopher all the things!"
PRINT "Start a Gopher server on port 70"
PRINT ""

REM Routes connect the incoming Gopher requests to functions
REM that you define. Each function should return a Gopher Menu
REM with the data you want filled in.
REM The first string is the incoming selector (which defaults to "")
REM The second string is the function to call
Gopher.AddRoute ("", "GOPHER_MAIN")
Gopher.AddRoute ("/time", "GOPHER_TIME")

REM Start the Gopher server
REM The Gopher server will be available via DNS-SD using the name in Gopher
Start.
status = Gopher.Start("motd")
PRINT status

FOREVER WAIT

REM Data passed in by the user.
FUNCTION GOPHER_MAIN(selector, ids, search)
    Screen.ClearLines (4,10)
    PRINT "Selector " + selector

    REM Create the menu to return
    REM The parameters are
    REM 1. type. "i" is information and "1" is a menu (directory)
    REM 2. user. A string that is displayed to the user
    REM 3. selector. For type "1", the selector (route) that
    REM determines which function is called
    REM 4. host. The host to go to for the menu. Default is the current host
    REM 5. port. The port number to go to. Default is the current port
    REM 6. options. Common options are INLINE and TITLE
    REM
    REM Be sure to return the menu from your function
    menu = Gopher.NewMenu()
    menu.Add ("i", "Hello gopher world!")
    menu.Add ("1", "Get current time (not inline)", "/time")
    menu.Add ("1", "Get current time (inlined)", "/time", "", "", "INLINE")
    menu.Add ("i", "<<unknown current time>>")
    menu.Add ("1", "GOTO MAIN", "")

RETURN menu

REM Called when a Gopher program asks for a /time selector
REM selector will be the Gopher selector (/time) based on Gopher.AddRoute
REM selector will be /time
REM ids will be blank, as will search
FUNCTION GOPHER_TIME(selector, ids, search)
    now = DateTime.GetNow()

    Screen.ClearLines (4,10)
    PRINT "Selector " + selector + " at " + now.Iso8601

    REM Create the menu to return
    REM Type "i" is information
    REM Type "1" is a new directory (menu)
    REM The Time menu is sometimes called as inline,
    REM so make sure that the first line returned
```

```
REM is useful information.
menu = Gopher.NewMenu()
menu.Add ("i", "ISO: " + now.Iso8601)
menu.Add ("i", "RFC 1123: " + now.Rfc1123)
menu.Add ("i", "Seconds: " + now.ASTotalSeconds)
menu.Add ("i", "")
menu.Add ("1", "GOTO MAIN", "")
```

```
RETURN menu
```

## 40.11.2 GOPHER of things

Demonstrates how a single program can connect to two different IOT devices via Bluetooth (TI SensorTag and a Bluetooth-enable light) and enable the user to control the devices using a GOPHER menu system

```
CLS
PRINT "Gopher of things!"
PRINT "Start a Gopher server on port 70 connected to a DOTI
device"
PRINT " "

REM Make an array of devices to control
device = Bluetooth.PickDevicesName("*Dotti")

IF (device.IsError)
    PRINT "No device was picked"
    STOP
END IF

REM Make a global array of the devices
DIM devices()
devices.Add (device.As ("DOTI"))

REM Routes connect the incoming Gopher requests to functions
REM that you define. Each function should return a Gopher Menu
REM with the data you want filled in.
REM The first string is the incoming selector (which defaults to
"")
REM The selector can include values ("{id}") which match anything
anything.
REM The "ids" array will include all of the values.
REM The second string is the function to call
Gopher.AddRoute ("", "GOPHER_MAIN")
Gopher.AddRoute ("/setRGB/{id}/{red}/{green}/{blue}",
"GOPHER_LIGHT")

REM List all of the lights

REM Start the Gopher server
REM The Gopher server will be available via DNS-SD using the name
in Gopher Start.
status = Gopher.Start("dotti-gopher")
PRINT status

FOREVER WAIT

REM Create the main menu. This will list each of the devices that
will be
REM controlled via the GOPHER program
FUNCTION GOPHER_MAIN(selector, ids, search)
    Screen.ClearLines (4,10)
    GLOBAL devices

    REM Create a menu with all devices to control
```

```

REM Be sure to return the menu from your function
menu = Gopher.NewMenu()
menu.Add ("i", "DOTTI CONTROL", "TITLE")
menu.Add ("i", " ")

FOR i = 1 TO devices.Count
    REM For each device, make an info line
    REM and four color set lines
    dotti = devices[i]
    menu.Add ("i", "Device " + i + " Name=" + dotti.GetName())
    menu.Add ("1", "RED", "/setRGB/" + i + "/255/0/0")
    menu.Add ("1", "GREEN", "/setRGB/" + i + "/0/255/0")
    menu.Add ("1", "BLUE", "/setRGB/" + i + "/0/0/255")
    menu.Add ("1", "OFF", "/setRGB/" + i + "/0/0/0")
NEXT i
RETURN menu

REM Handle a set light request
REM The route was /setRGB/{id}/{red}/{green}/{blue} so the
REM ids array will have 4 values which you can pull out by name
REM (e.g., ids.red is the {red} value and is expected to be a
number 0..255)
FUNCTION GOPHER_LIGHT(selector, ids, search)
    now = DateTime.GetNow()

    REM For debugging and tracing, print out some useful values.
    Screen.ClearLines (4,10)
    PRINT "Device ", ids.id
    PRINT "R", ids.red
    PRINT "G", ids.green
    PRINT "B", ids.blue

    REM Use the id value to pick the right device
    REM and then call the Dotti.SetPanel method to set
    REM the r/g/b values
    GLOBAL devices
    dotti = devices[ids.id]
    dotti.SetPanel (ids.red, ids.green, ids.blue)

    REM Create the menu to return. Add in handy debugging values.
    menu = Gopher.NewMenu()
    menu.Add ("i", "Set DOTTI panel " + dotti.GetName())
    menu.Add ("i", "R=" + ids.red + " G=" + ids.green + " B=" + ids.blue)
    menu.Add ("1", "GOTO MAIN", "")

RETURN menu

```

### 40.11.3 Gopher program with changing responses

This GOPHER program demonstrates how you can provide access to selectable data. The GOPHER\_MESSAGE function returns a GOPHER menu with one of four different values depending on the selector that's passed in. The GOPHER\_SEARCH function returns data based on the search string (it returns population statistics for different cities)

```

CLS
PRINT "Gopher all the things!"
PRINT "Start a Gopher server on port 70"
PRINT ""

```

```
Gopher.AddRoute ("", "GOPHER_MAIN")
Gopher.AddRoute ("/message/{msg}", "GOPHER_MESSAGE")
Gopher.AddRoute ("/search/{item}", "GOPHER_SEARCH")

REM Start the Gopher server
REM The Gopher server will be available via DNS-SD using the name
in Gopher Start.
status = Gopher.Start("message")
PRINT status
startTime = DateTime.GetNow()

FOREVER WAIT

REM Data passed in by the user.
FUNCTION GOPHER_MAIN(selector, ids, search)
    Screen.ClearLines (4,10)
    PRINT "Selector " + selector

    REM Create the menu to return
    menu = Gopher.NewMenu()
    menu.Add ("i", "Messages from GOPHER")
    menu.Add ("1", "Uptime", "/message/uptime")
    menu.Add ("1", "Current time", "/message/time")
    menu.Add ("1", "Current date", "/message/date")
    menu.Add ("1", "Version", "/message/version")
    menu.Add ("7", "Search", "/search/main")

    menu.Add ("1", "GOTO MAIN", "")
RETURN menu

REM Called when a Gopher program asks for a /message selector
FUNCTION GOPHER_MESSAGE(selector, ids, search)
    now = DateTime.GetNow()
    GLOBAL startTime
    delta = now.Subtract (startTime)

    Screen.ClearLines (4,10)
    PRINT "Selector " + selector + " message " + ids.msg

    REM Create the menu to return
    menu = Gopher.NewMenu()
    menu.Add ("i", "Information about the server")
    IF (ids.msg = "uptime") THEN menu.Add ("i", "Uptime: " + delta)
    IF (ids.msg = "time") THEN menu.Add ("i", "Time: " + now.Time)
    IF (ids.msg = "date") THEN menu.Add ("i", "Date: " + now.Date)
    IF (ids.msg = "version") THEN menu.Add ("i", "Version: " +
System.Version)
    menu.Add ("i", "")
    menu.Add ("1", "GOTO MAIN", "")

RETURN menu

REM Called when a Gopher program asks for a /search selector
FUNCTION GOPHER_SEARCH(selector, ids, search)
    now = DateTime.GetNow()

    Screen.ClearLines (4,10)
    PRINT "Search " + selector + " item " + ids.item
    PRINT "Search for: " + search
    cities = Data.GetLocations (search)
    FOR i = 1 TO cities.Count
        PRINT cities[i]
    NEXT i
    REM Create the menu to return
    menu = Gopher.NewMenu()
    menu.Add ("i", "Search results")
    FOR i = 1 TO cities.Count
        city = cities[i]
```

```
    menu.Add ("i", city.FullName + " " + city.Population)
NEXT i
menu.Add ("i", "")

menu.Add ("1", "GOTO MAIN", "")

RETURN menu
```

## 40.12 EX: REAL ESTATE

Convert square feet to acres and more

### 40.12.1 Acres to square feet

Converts acres from the calculator display into square feet

```
value = Calculator.Value
retval=value * 43560
Calculator.Message = "Converted " + value + " acres into square feet"
STOP retval
```

### 40.12.2 Debt to Income calculations

Given two numbers -- the borrower's yearly income and the bank's income limit (e.g., 31 for 31% allowed for housing), calculates the allowed amount per month for housing.

```
income = INPUT DEFAULT 100000 PROMPT "What is the person's
yearly income"
monthlyIncome = income / 12

housingPercent = INPUT DEFAULT 31 PROMPT "What is the allowed
housing debt to income ratio?"
maxPercent = INPUT DEFAULT 43 PROMPT "What is the allowed debt
to income ratio? "

allowedMonthlyHousing = INT (monthlyIncome *
housingPercent/100)
allowedMonthlyTotal = INT (monthlyIncome * maxPercent/100)
```

**CLS**

```
PRINT "Income per month="; INT(monthlyIncome)
PRINT "Housing per month=";allowedMonthlyHousing
PRINT "Total monthly debt=";allowedMonthlyTotal
PRINT "Other debt=";allowedMonthlyTotal-allowedMonthlyHousing
Calculator.Message = "For year income of " + income + " housing per
month is " + allowedMonthlyHousing
STOP allowedMonthlyHousing
```

### 40.12.3 Minimum and Maximum density

Demonstrates one way to calculate the minimum and maximum number of units that can be built on a lot given its size in acres. The rules roughly match those of Redmond, WA for residential neighborhoods (Redmond code 20C.30.25) as of 2015. You will need to supply the R type (e.g., 1 for R1, 4 for R4).

```
R = INPUT DEFAULT 6 PROMPT "What is the R type zoning district?
Enter .2 for type RA-5"
grossAcres = INPUT DEFAULT 2 PROMPT "What is the gross site area
(acres)?"
netAcres = INPUT DEFAULT grossAcres PROMPT "What is the net
buildable area (acres)?"
```

REM Sample: R=6 grossAcres=2 netAcres=1.5 result in minimum 7  
maximum 12 house

```
allowed = Math.Round (RtoAllowedDensity(R) * grossAcres)
minimum = Math.Round (RtoMinimumDensity(R) *
RtoAllowedDensity(R) * netAcres)
```

```
REM How much more land do you need to build 1 more house?
start = RtoAllowedDensity(R) * grossAcres
fraction = start - Math.Floor (start)
IF fraction >= .5 THEN next = Math.Ceiling(start) + .5
IF fraction < .5 THEN next = Math.Floor(start) + .5
delta = next - start
```

```
deltaAcres = delta / RtoAllowedDensity(R)
```

```
CLS
```

```
PRINT "You can build up to "; allowed; " houses"  
PRINT "You must build at least "; minimum; " houses"  
PRINT "You need "; deltaAcres; " more acres to build 1 more house"  
STOP
```

```
REM The R number exactly matches the allowed density for all values  
REM Except RA-5 which must be entered as .2
```

```
FUNCTION RtoAllowedDensity (R)  
RETURN R
```

```
REM There are three minimum density sizes in Redmond
```

```
FUNCTION RtoMinimumDensity(R)  
IF R < 8 THEN RETURN .8  
IF R < 18 THEN RETURN .75  
RETURN .65
```

#### 40.12.4 Rectangle in feet to acres

Given a lot size in feet, calculates the lot size in acres

```
length = INPUT DEFAULT 80 PROMPT "Enter the first length in feet"  
width = INPUT DEFAULT 11 PROMPT "Enter the second length in feet"  
sqfeet = length * width  
acres=sqfeet / 43560  
Calculator.Message = "Lot " + length + "x" + width + " is " + acres +  
" acres"  
STOP retval
```

#### 40.12.5 Square feet to acres

Converts the current values in the calculator from square feet to acres.

```
value = Calculator.Value
```

```
retval=value / 43560
```

```
Calculator.Message = "Converted " + value + " square feet into acres"
```

```
STOP retval
```

## 40.13 EX: SPACE AND ASTRONOMY

Programs for astronomy and space

### 40.13.1 Arc Length

A COGO program to calculate an arc length or a circle given the radius and the angle (in degrees)

```
REM 3959 is the radius of the earth in miles
```

```
TEST()
```

```
radius = INPUT DEFAULT 3959 PROMPT "Radius of the circle"
```

```
degrees = INPUT DEFAULT 45 PROMPT "Angle in degrees"
```

```
arc = ArcLength(degrees, radians)
```

```
Calculator.Message = "Given radius=" + radius + " and angle=" +  
angle + " arc is " + arc
```

```
STOP arc
```

```
FUNCTION ArcLength(degrees, radius)
```

```
radians = Math.DtoR (degrees)
```

```
circum = Math.PI * 2 * radius
```

```
arc = circum * radians / (2 *Math.PI)
```

```
RETURN arc
```

```
FUNCTION TestOne (degrees, radius, expected)
```

```
actual = ArcLength (degrees, radius)
```

```
IF (actual ≈ expected) THEN RETURN 0
```

```
PRINT "ERROR; ArcLength ";h
```

```
PRINT "Expected "; expected
```

```
PRINT "Actual "; actual
```

```
PRINT "Difference "; actual-expected
```

```
RETURN 1
```

```
FUNCTION TEST ()
```

```
nerror = 0
```

```
nerror = nerror + TestOne (3959, 45, 3109.391)
```

```
nerror = nerror + TestOne(10, 90, 15.707963)
```

```
nerror = nerror + TestOne(0, 90, 0)
```

```
nerror = nerror + TestOne(10, 0, 0)
```

```
IF (nerror > 0) THEN PRINT "HORIZON NERROR=";nerror
```

```
IF (nerror > 0) THEN PAPER RED
```

```
RETURN nerror
```

### 40.13.2 AU to Meters

Converts a distance in AU (Astronomical units) to a distance in meters

```
IMPORT FUNCTIONS FROM "Conversion Library"
```

```
from = Calculator.Value
```

```
m = ConvertToMeters(from, "au")
```

```
Calculator.Message = "Convert " + from + " au into " + m + "  
meters"
```

```
STOP m
```

### 40.13.3 Conversion Library

A set of functions to convert between AU and kilometer and between Parsecs, Lightyears and Meters

```
FUNCTION ConvertToMeters(distance, units)
```

```
IF units = "au" THEN RETURN distance * 149597870700
```

```
IF units = "earthradius" THEN RETURN distance * 6371000
```

```
IF units = "lightsecond" THEN RETURN distance * 299792458
```

```
IF units = "lightyear" THEN RETURN distance * 9.4605284E15
```

```
IF units = "parsec" THEN RETURN distance * 3.08567758E16
```

```
CONSOLE "ERROR: Astronomy Conversion library: Unknown units " +  
units
```

```
RETURN Math.NaN
```

```
CLS BLACK
```

```
TEST()
```

```
FUNCTION TestOne (distance, units, expected)
actual = ConvertToMeters (distance, units)
IF (actual ≈ expected) THEN RETURN 0
IF (Math.IsNaN(expected) AND Math.IsNaN(actual)) THEN RETURN 0
PRINT "ERROR; ConvertToMeter "; distance; " "; units
PRINT "Expected "; expected
PRINT "Actual "; actual
PRINT "Difference "; actual-expected
RETURN 1
```

```
FUNCTION TEST ()
```

```
nerror = 0
```

```
REM Test AU with data from Mercury and Neptune
```

```
nerror = nerror + TestOne (.387, "au", 5.7894E10)
```

```
nerror = nerror + TestOne (30.06, "au", 4.4969E12)
```

```
nerror = nerror + TestOne (1, "lightsecond", 299792.458E3)
```

```
REM Wikipedia says the distance Earth to Moon is 1.282
```

```
nerror = nerror + TestOne (1.28222, "lightsecond", 384400E3)
```

```
REM data from Bing.com
```

```
nerror = nerror + TestOne (1, "lightyear", 9.4605284E15)
```

```
nerror = nerror + TestOne (1, "earthradius", 6371E3)
```

```
nerror = nerror + TestOne (1, "parsec", 3.08567758E16)
```

```
nerror = nerror + TestOne (1, "NOSUCHUNIT", Math.NaN)
```

```
IF (nerror > 0) THEN PRINT "Astronomical Conversion"
```

```
NERROR=";nerror
```

```
IF (nerror > 0) THEN PAPER RED
```

```
RETURN nerror
```

**40.13.4 Distance to horizon**

Calculates the distance to the horizon in miles given a height above the Earth in feet.

```
h = Calculator.Value  
TEST ()  
d = Distance (h)  
Calculator.Message = "Given a height of " + h + " feet, the distance to  
horizon in miles is " + d  
STOP d  
  
FUNCTION Distance (height)  
d = 1.22 * SQR(height)  
RETURN d  
  
FUNCTION TestOne (h, expected)  
actual = Distance(h)  
IF (actual ≈ expected) THEN RETURN 0  
PRINT "ERROR; DistanceToHorizon ";h  
PRINT "Expected "; expected  
PRINT "Actual "; actual  
PRINT "Difference "; actual-expected  
RETURN 1  
  
FUNCTION TEST ()  
nerror = 0  
nerror = nerror + TestOne (100, 12.2)  
nerror = nerror + TestOne(22841, 184.382)  
PRINT "HORIZON NERROR=";nerror  
IF (nerror > 0) THEN PAPER RED  
RETURN nerror
```

**40.13.5 Lightyears to Parsecs**

Converts a distance in parsecs to a distance in light-years

**IMPORT FUNCTIONS FROM "Conversion Library"**

```
from = Calculator.Value  
m = ConvertToMeters(from, "lightyear")  
inv = ConvertToMeters(1, "parsec")  
to = m / inv  
Calculator.Message = "Convert " + from + " into " + to + " parscecs"  
STOP to
```

**40.13.6 Meters to AU**

Converts a distance in meters to a distance in AU (Astronomical units)

**IMPORT FUNCTIONS FROM "Conversion Library"**

```
from = Calculator.Value  
inv = ConvertToMeters(1, "au")  
to = from / inv  
Calculator.Message = "Convert " + from + " into " + to + " au"  
STOP to
```

**40.13.7 Parsecs to Lightyears**

Converts a distance in parsecs to a distance in light-years

**IMPORT FUNCTIONS FROM "Conversion Library"**

```
from = Calculator.Value  
m = ConvertToMeters(from, "parsecIES!")  
m = ConvertToMeters(from, "parsec")  
inv = ConvertToMeters(1, "lightyear")  
to = m / inv  
Calculator.Message = "Convert " + from + " into " + to + " light  
years"  
STOP to
```

## 40.13.8 Rocket Equation

The Tsiolkovsky rocket equation will tell you how much fuel you have to burn in order to achieve some change in velocity (delta-v). You have to provide the starting rocket weight (including fuel) and the rocket effective exhaust velocity. The Space Shuttle effective exhaust velocity is 4,400 m/s.

```
REM From https://en.wikipedia.org/wiki/Tsiolkovsky_rocket_equation
```

```
REM  $\Delta v = ve \ln(m_0 / m_1)$ 
```

```
REM  $m_0$ =initial mass
```

```
REM  $m_1$ =final mass (after fuel is burnt)
```

```
REM  $ve$  = effective exhaust velocity (about 4400 for Space Shuttle  
main engines)
```

```
REM Solve the equation for  $m_1$  to return  $(m_0 - m_1)$ , the amount of  
propellant to burn
```

```
REM  $\Delta v = ve \ln(m_0 / m_1)$ 
```

```
REM  $\Delta v / ve = \ln(m_0/m_1)$ 
```

```
REM  $e^{\Delta v / ve} = m_0/m_1$ 
```

```
REM  $m_0 / e^{\Delta v / ve} = m_1$ 
```

```
REM fuelburned =  $m_0 - (m_0 / e^{\Delta v / ve})$ 
```

```
m0 = INPUT DEFAULT 727 PROMPT "Starting mass of the rocket (any  
units)"
```

```
REM Sample specific impulse is from the Centaur rocket
```

```
isp = INPUT DEFAULT 450.5 PROMPT "Specific Impulse (in seconds)"
```

```
REM delta-v of about 6,900,000 is (approximately) the amount  
needed to get to low earth orbit
```

```
deltav = INPUT DEFAULT 5000 PROMPT "What delta-v do you need?  
(in meters/second)"
```

```
CLS GREEN
```

```
ve = SpecificImpulseToEffectiveVelocity(isp)
```

```
fuelburned = FuelBurned (m0, deltav, ve)
m1 = m0-fuelburned
```

```
PRINT "Starting rocket mass="; m0
PRINT "Specific Impulse="; isp
PRINT "Exhaust Velocity="; ve
PRINT "Change in velocity=";deltav; "(m/s)"
PRINT "Fuel burned=";fuelburned
PRINT "Final rocket mass=";m1
IF (m1 > 0) THEN GOTO 10
PRINT "Ran out of fuel!"
PAPER RED
10 REM done
```

```
Calculator.Message = "You used " + fuelburned + "fuel"
STOP fuelburned
```

```
FUNCTION SpecificImpulseToEffectiveVelocity (Isp)
g0 = 9.81
Ve = g0 * Isp
RETURN Ve
```

```
FUNCTION FuelBurned (m0, deltav, ve)
ratio = 1 / EXP (deltav / ve)
m1 = m0 * ratio
fuelburned = m0 - m1
RETURN fuelburned
```

## 40.14 EX: STATISTICS

Sample programs for statistics

### 40.14.1 Finite Population Correction

Corrects the Margin of Error calcations when drawing from a finite instead of infinit population.

**IMPORT FUNCTIONS FROM "Sample Size Library"**

```
population = INPUT DEFAULT 1000000 PROMPT "Enter the actual  
population"  
sample = INPUT DEFAULT 1000 PROMPT "Enter the number of  
samples"  
fpc = FPC (sample, population)  
Calculator.Message = "Finite Population Correction for  
population=" + population  
STOP fpc
```

**40.14.2 Margin of Error**

Calculates the margin of error for sampling an infinite population given a sample size

**IMPORT FUNCTIONS FROM "Sample Size Library"**

```
z = GetZ()  
n = INPUT DEFAULT 1000 PROMPT "How many samples will you take?"  
stddev = 0.5  
REM 0.5 is a very conservative approach and applies a derate factor of  
.25  
REM choosing .1 would result in a derate of 0.09 which is not wildly  
different  
  
me = MarginOfError (z, n, stddev)  
Calculator.Message = "Margin of error for sample size "+n  
STOP me
```

**40.14.3 Pfail**

Returns the probability of failure by time t given an MTBF (mean time to failure) value.

REM Note: MTBF is 1/lambda

```
TEST()
```

```
MTBF = INPUT DEFAULT 10 PROMPT "MTBF (Mean Time Before Failure)"
```

```
T = INPUT DEFAULT 5 PROMPT "Time to calculate from"
```

```
P = Pfail (T, MTBF)
```

```
Calculator.Message = "Pfailure by time T given MTBF"
```

```
STOP P
```

```
FUNCTION Pfail (T, MTBF)
```

```
P = 1 - Math.E ** ( - (T / MTBF))
```

```
RETURN P
```

```
FUNCTION TestOne (T, MTBF, Expected)
```

```
Actual = Pfail (T, MTBF)
```

```
IF (Actual ≈ Expected) THEN RETURN 0
```

```
PRINT "Pfail: T=";T;" MTBF="; MTBF
```

```
PRINT "Actual=";Actual
```

```
PRINT "Expected=";Expected
```

```
RETURN 1
```

```
FUNCTION TEST()
```

```
nerror = 0
```

```
nerror = nerror + TestOne (43800, 250000, 0.16071)
```

```
RETURN nerror
```

#### 40.14.4 Sample Size

Calculates the sample size required to meet a desired margin of error given a confidence limit.

```
REM Calculates a required Sample Size
```

```
REM You have to enter a Confidence (90, 95 or 99%)
```

```
REM You have to enter a margin of error (e.g., 3 to mean 3%)
```

```
REM
```

```
REM The code assume an infinite populat. Use the FPC (Finite  
Population Correction)
```

```
REM if the population size is smaller.
```

```
REM
```

```
REM The code assumes a population stddev of 0.5; this is the most  
REM conservative assumption.
```

```
REM
```

```
IMPORT FUNCTIONS FROM "Sample Size Library"
```

```
PRINT "Sample Size"
```

```
TEST()
```

```
z = GetZ()
```

```
me = INPUT DEFAULT 5 PROMPT "What is your required margin of  
error in percent?"
```

```
me = me / 100
```

```
stddev = 0.5
```

```
REM 0.5 is a very conservative approach and applies a derate factor of  
.25
```

```
REM chosing .1 would result in a derate of 0.09 which is not wildly  
different
```

```
n = SampleSize(z, me, stddev)
```

```
STOP n
```

## 40.14.5 Sample Size Library

Useful functions for calculating sample size

```
REM A set of functions and tests for sample sizes
```

```
FUNCTION MarginOfError(z, n, stddev)
me = SQR ((z**2 * stddev * (1-stddev)) / n)
RETURN me
```

```
FUNCTION SampleSize(z, me, stddev)
n = z**2 * stddev * (1-stddev) / me**2
RETURN n
```

```
FUNCTION FPC(sample, population)
REM Finite Population Correction; corrects the Margin of Error
REM based on drawing from a finite (instead of infinite) population
ratio = (population - sample) / (population - 1)
fpc = SQR(ratio)
RETURN fpc
```

```
FUNCTION GetZ()
10 confidence = INPUT DEFAULT 95 PROMPT "Required confidence
level (one of 90, 95 or 99)"
z = Z(confidence)
IF (z = 0) THEN GOTO 10
RETURN z
```

```
FUNCTION Z(confidence)
IF (confidence = 90) THEN RETURN 1.645
IF (confidence = 95) THEN RETURN 1.96
IF (confidence = 99) THEN RETURN 2.576
RETURN 0
```

```
FUNCTION TestOne (confidence, n, me, stddev)
nerror = 0
z = Z(confidence)
nactual = SampleSize (z, me, stddev)
IF (nactual ≈ n) THEN GOTO 20
PRINT "ERROR: SampleSize (pt1)"
PRINT "z=";z
PRINT "me=";me
PRINT "expected n=";n
PRINT "actual n=";nactual

20 meactual = MarginOfError (z, n, stddev)
IF (meactual ≈ me) THEN GOTO 30
PRINT "ERROR: SampleSize (2)"
PRINT "z=";z
PRINT "n=";n
PRINT "expected me=";me
PRINT "actual me=";meactual

30 REM all done
RETURN nerror
```

```
FUNCTION TestFPC(sample, population, fpc)
nerror = 0
actual = FPC (sample, population)
IF (actual ≈ fpc) THEN RETURN 0
PRINT "ERROR: FPC"
PRINT "sample=";sample
```

```
PRINT "population=";population
PRINT "fpc=";fpc;" actual=";actual
RETURN nerror
```

```
FUNCTION QuickTest()
nerror = 0
nerror = nerror + TestFPC (1000, 1000000, 0.99950037)
nerror = nerror + TestFPC (20, 50, 0.78246)
nerror = nerror + TestOne (99, 1727.34, 0.030990321, 0.5)
nerror = nerror + TestOne (95, 1000, 0.030990321, 0.5)
nerror = nerror + TestOne (90, 704.4, 0.030990321, 0.5)
nerror = nerror + TestOne (95, 1000, 0.026838405, 0.25)
RETURN nerror
```

```
FUNCTION TEST()
nerror = QuickTest()
RETURN nerror
```

```
REM RUN the library in order to test it!
```

```
PRINT "Testing Sample Size Library"
TEST()
```

# INDEX TO BEST CALCULATOR MANUAL

---

- (subtract), 117
- (subtract, BASIC), 116
- (subtract, Calculator), 25
- !
- (factorial) key, 37
- M (store)
  - In Calculator Memory Page, 47
- M (To Memory) key, 28
- % (percent) key, 32
- & (and) key, 54
- ( ) (parentheses, Calculator), 26
- ( ) (parenthesis)
  - when calling a function, required, 115
- ( ) (parenthesis, BASIC)
  - in expressions, 115
- \*(multiply), 117
- \*\* (raise to the power), 117
- / (divide), 117
- [ ] (square brackets)
  - in expressions, 115
- ^ (xor) key, 54
- ^[ Indent, 104
- ^] Un-indent, 104
- ^F FIND, 103
- ^G GO TO, 103
- | (or) key, 54
- ~ (inverse) key, 53
- + (add), 117
- + (add, Calculator), 25
- < (less than), 107, 118
- <= (less than or equal), 107, 118
- <> (not equal), 107, 118
- = (equal), 107, 118
- = (equals, Calculator), 25
- > (greater than), 107, 118
- >= (greater than or equal), 107, 118
- ± (change sign) key, 30
- ≪ (shift left) key, 56
- ≪+○ (rotate left) key, 56
- ≫ (shift right) key, 56
- ≫+○ (rotate right) key, 56
- × (multiply, Calcualtor), 25
- ÷ (divide, Calculator), 25
- √ (squarae root), 116
- √ (square root) key, 30
- ³√ (cube root), 116
- ³√x key, 37
- ⁴√ (fourth root), 116
- ≈ (approximately equal), 118
- ≢ (Not approximately equal to), 118
- (Delete) key, 26
- ☰ (word) key, 55
- ☰☰ (dword) key, 55
- (byte) key, 55
- b, in PLAY command, 152
- #, in PLAY command, 152
- 1/x (Inverse) key, 30
- ² (square), 116
- 2's complement key, 53
- 2541, 284
- ³ (cube), 116
- ⁴ (fourth power), 116
- 7-segment font, 78
- ∞ (Infinity), 110

∞ (infinity) key, 45  
∞, Infinity, 131  
ABS (BASIC), 123  
Abs key, 39  
AccelerometerSetup, 284, 288  
ACS (BASIC), 122  
Add  
    array, 141  
    DateTime, 167  
Add, array, 139  
AddPoints, Polygon, 207  
AddRoute  
    Gopher, 176  
AddRow  
    array, 142  
AddRow, array, 139  
Admin1, Data field, 163  
AlarmSetting, 269  
Algebraic Entry, 25  
Algebraic Entry System with  
    Hierarchy, 26  
Algebraic Operating System, 26  
Alignment, 77  
Alignment, Text, 210  
altitude, 216  
altitude, Infineon, 261  
AnalogWrite, Ardudroid, 249  
Analyze, Sensor.Camera, 213  
AND, 118  
And (&) key, 54  
Antique Carriage font, 78  
Antique Segment font, 78  
Appearance, 77  
AppendLine  
    File, 170  
AppendPicker  
    File, 170  
AppendText  
    File, 170  
Arc, 205  
Ardudroid, 249  
Arduino, 248, 282  
array, 137  
As  
    Bluetooth, device.As(), 234  
ASC (CODE), 125  
ASCII Table, 73  
ASN (BASIC), 122  
ASSERT, Testing, 107  
*assignment*, 151  
AsTotalSeconds  
    DateTime, 169  
ATN (BASIC), 122  
automatic graph, 196  
Average  
    Example BASIC program, 138  
B# (count bits) key, 53  
BarometerSetup, SensorTag  
    2541, 285, 288  
BEEP, 133  
beLight, 254  
bin key, 51  
Binary numbers, 51  
bind a program to a key, 94  
Bluetooth, 228  
Bluetooth object, 232  
Bluetooth.Devices, 233  
Bluetooth.Watch, 236  
Border, 195  
Boxplots, 67  
Button, 205  
ButtonSetup, SensorTag 2541,  
    285, 288  
byte (ﷺ) key, 55  
C (clear) key, 26  
c (speed of light) key, 45  
cake, chocolate, 307  
Calculator.Message, 162  
Calculator.Value, 162  
Calendar calculations, 49

## Guide to Using Best Calculator

- CALL, 134
- callback function, Gopher, 174
- callbacks, Bluetooth, 243
- Camera, Sensor, 212
- Carriage return, 131
- CC2540T, 254
- CE (clear entry) key, 26
- Ceil key, 39
- Chain calculations, 25
- ChangeMode, 256, 270
- characteristic, Bluetooth, 240
- chocolate cake, 307
- CHR, 125
- Circle
  - Graphics, 205
- Classical Statistics, 61
- Clear, array, 140
- Clear, Graphics Screen, 195
- ClearGoTo
  - Graphics, 207
- CLS, 134
  - clearing the screen manually, 97
- CODE, 125
- color
  - String.Escape, 220
- Comment (using REM), 159
- Compass, Sensor, 215
- Console, 105
  - CONSOLE command, 315
  - CONSOLE Command, 136
  - display or hide (toggle), 106
  - DUMP command, 315
- Constants (Calculator), 45
- Conversions
  - Between bases, 51
  - BTUs, 70
  - Bushels, 72
  - Calories, 70
  - Celsius, 71
- Centimeters, 71
- Cups, 72
- degrees to radians, 33
- Donuts, 70
- Ergs, 70
- Fahrenheit, 71
- Feet, 71
- Gallons, 72
- Grains, 71
- Grams, 71
- Inches, 70
- Joules, 70
- Kelvin, 71
- Kilograms, 71
- Kilometers, 71
- Kilowatt-Hours, 70
- Liters, 72
- Long tons, 71
- Maund, 72
- Meters, 71
- Miles, 71
- MMT, 71
- Ounces, 71
- Pecks, 72
- Pints, 72
- Pounds, 71
- Quarts, 72
- radians to degrees, 33
- Rankine, 71
- Sér, 71
- Short tons, 71
- Therms, 70
- Tolä, 71
- Tonnes, 71
- Troy Ounces, 71
- Troy Pounds, 71
- Yards, 71
- Correlation coefficient, 65
- COS (BASIC), 122
- Cos key, 33

Count, 61  
array, 140  
count bits (B#) key, 53  
Country, Data field, 163  
csv  
    String.Escape, 220  
    String.Parse, 222  
CX, Graphics, 202  
CXD, Graphics, 202  
CY, Graphics, 202  
CYD, Graphics, 202  
d→r key, 33  
DATA, 158  
data boxes, 59  
Data extension, 163  
Data, Graphics, 202  
Date  
    DateTime, 168  
Date Calculations, 49  
DateTime Extension, 165  
David (SPEAK VOICE), 159  
Day  
    DateTime, 168  
DayOfWeek  
    DateTime, 168  
DayOfYear  
    DateTime, 168  
dec key, 51  
Decimal numbers, 51  
degree and radians  
    In Calculator, 33  
degrees key, 33  
degrees of freedom, 66  
Desktop Shortcut, 75  
df, 66  
DigitalWrite, Ardudroid, 249  
DIM, 137  
Direct Algebraic Logic, 26  
DOTTI, 256  
double  
numbers stored as doubles,  
    110  
DPS310, 261  
DUMP, 142  
duration (BEEP), 133  
dword (DWORD) key, 55  
e key, 45  
Eddystone, 236  
Eddystone-URL, 237  
EE (scientific notation) key, 31  
Elevation, Data field, 163  
Ellipse, 206  
ELSE, 148  
END, 160  
Errors, System, 226  
Escape  
    String, 220  
Escape (stop program) key, 105  
EXP (BASIC), 122  
exponent, BASIC, 184  
exponential notation (BASIC),  
    110  
exponential notation  
    (Calculator), 31  
expressions, 114  
F5 (run program) key, 105  
Factorial (Calculator), 37  
fff0, 258  
fff0, NOTTI, 268, 271  
fff3, 258  
Fft, 185  
File Extension, 170  
Fill, array, 140  
Find, 103  
First  
    RemoveAlgorithm, 141  
Fixed character size screen, 105  
floats, 110  
Floor key, 39  
Flux light, 263

FolderBasic  
    System, 226

FolderTemporary  
    System, 226

Font size  
    indicator, 106  
    larger, 106  
    smaller, 106

fontsize, Text, 210

FOR..NEXT loops, 143

FOREVER [WAIT|STOP], 146

Formatting (Calculator), 42

Fourier transform, 185

Frac key, 39

FullName, Data field, 163

FullScreenGraphics, 195

FUNCTION, 146

GeoNameId, Data field, 163

Get  
    Bluetooth.Devices, 233  
    Http, 178

GetAccelerometer, Hexiwear,  
    260

GetCalories, Hexiwear, 260

GetFirmwareRevision,  
    Hexiwear, 259

GetGyroscope, Hexiwear, 260

GetHeart, Hexiwear, 260

GetHumidity, Hexiwear, 260

GetLight, Hexiwear, 260

GetLocations  
    Data, 163

GetMagnetometer, Hexiwear,  
    260

GetManufacturerName,  
    Hexiwear, 259

GetMode, 260

GetNow  
    DateTime, 166

GetNowUtc

    DateTime, 166

GetPower, Hexiwear, 260

GetPressure, Hexiwear, 260

GetSteps, Hexiwear, 260

GetTemperature, Hexiwear,  
    260

GLOBAL, 147

g<sub>n</sub> (gravitational constant) key,  
    45

Go to, 103

Gopher and Gopher-of-Things  
    example, 413

Gopher Extension, 173

GOSUB, 147

GOTO, 148

Graphics  
    Screen, 195

GraphY  
    Graphics, 196, 197

Greek letters in variable names,  
    112

Gregorian Calendar, 49

GUID, 240

GX, Screen, 193

GY, Screen, 193

GyroscopeSetup, SensorTag  
    2541, 285

Happy Birthday, 307

HasKey(name), array, 140

HC-06, Bluetooth, 282

Hebrew Calendar, 49

Herschel notation, 34

hex key, 51

Hex numbers, 51

Hexiwear, 259

Hijiri Calendar, 49

History of BASIC, 80

Hour  
    DateTime, 168

HourDecimal

DateTime, 168  
HTML, pretty-print, 130  
HTML, Pretty-print, 130  
Http Extension, 178  
HumiditySetup, SensorTag  
    2541, 285, 288  
IF - THEN, 148  
IFX\_NANOHUB, 261  
image, Screen.Graphics, 212  
IMPORT FUNCTIONS, 149  
import package, 93  
Inclinometer, Sensor, 215  
Indent lines, 104  
Infineon, 261  
infinite loop (stopping), 105  
infinity ( $\infty$ ), 110  
Init  
    Bluetooth, 234  
INPUT (operator, details), 121  
INPUT (operator, preferred),  
    119  
INPUT (statement, not  
    preferred), 150  
instruments (PLAY), 152  
INT (BASIC), 123  
Integer key, 39  
Integers, 110  
Intercept, 65  
Intersect, Graphics, 203  
Inverse Cos (calculator), 34  
Inverse Sin (calculator), 34  
Inverse Tan (calculator), 34  
InverseFft, 185  
IRSetup, SensorTag 2541, 285,  
    289  
IsError, 113  
isNaN, 114  
IsNumber, 113  
Iso8601  
    DateTime, 168  
IsObject, 113  
IsString, 113  
json  
    String.Escape, 221  
    String.Parse, 223  
Julian Calendar, 49  
Keyboard calculator button, 76  
latitude, 216  
LatitudeDD, Data field, 163  
LED1.set, 272  
LEDBlue, 263  
LEFT, 123  
LEN, 124  
LET, 151  
library  
    delete package, 96  
    export package, 96  
    how to display properties, 92  
    made up of packages, 91  
library  
    add package, 317  
Light, 254  
Light, Sensor, 215  
Line  
    Graphics, 205  
Line ending  
    \n, 109  
    \r, 109  
    \v, 109  
Linear Regression, 65, 68  
Linear Regression Chart, 65  
LineTo  
    Graphics, 207  
LittleBot, 282  
LN (BASIC), 122  
ln key, 35  
LoadScreenFromMemory, 257  
Location, 216  
log key, 35  
log<sub>2</sub> key, 35

Guide to Using Best Calculator  
Logarithms (Calculator), 35  
longitude, 216  
LongitudeDD, Data field, 163  
Looping (FOR..NEXT loops), 143  
lux  
    Puck.js, 273  
    Sensor, 215  
    TI SensorTag 1350, 289  
M- (memory subtract) key, 29  
M→ (recall)  
    In Calculator Memory Page, 47  
M→ (Recall) key, 28  
M-(memory subtract)  
    Calculator Memory Page, 47  
M+ (memory add)  
    In Calculator Memory Page, 47  
    key, 29  
MagicLight, 263  
magnetometer, 274  
MagnetometerSetup,  
    SensorTag 2541, 286  
Mark (SPEAK VOICE), 159  
Math Extension, 180  
Math.Abs, 181  
Math.Acos, 180  
Math.Asin, 180  
Math.Atan, 180  
Math.Atan2, 180  
Math.BitAnd, 183  
Math.BitNo, 183  
Math.BitOr, 183  
Math.Ceiling, 181  
Math.Cos, 180  
Math.Cosh, 180  
Math.DtoR, 180  
Math.E, 185  
Math.Exp, 183  
Math.Factorial, 184

Page | 415  
Math.Fft, 185  
Math.Floor, 181  
Math.Frac, 181  
Math.InverseFft, 185  
Math.NaN, 184  
Math.Log, 183  
Math.Log10, 184  
Math.Log2, 183  
Math.Max, 181  
Math.Min, 181  
Math.Mod, 181  
Math.NaN, 185  
Math.PI, 185  
Math.Pow, 184  
Math.Round, 182  
Math.RtoD, 180  
Math.Sign, 182  
Math.Sin, 180  
Math.Sinh, 180  
Math.Sqrt, 184  
Math.Tan, 181  
Math.Tanh, 181  
Math.Truncate, 182  
Max  
    array, 140  
MaxCount  
    array, 141  
MaxOf, 140  
mbientlab.com, 264  
Mean, 61, 140  
median, 63  
Memory  
    M+ (memory add) key, 29  
    Memory Recall (M→) key, 28  
    Memory Store (→M) key, 28  
    Memory Subtract (M-) key, 29  
    Naming memory, 47  
Memory Extension, 187  
Memory Page, 28, 47, 49

Memory.<constant\_name>, 188  
Memory.GetOrDefault, 188  
Memory.IsSet, 188  
Memory[<expression>], 187  
menu, Gopher, 174  
**MetaMotion**, **264**  
**MetaWear**, **264**  
Microphone, 217  
MID, 123  
mikroElektronika, 259  
Min  
    array, 140  
MinOf, 140  
Minute  
    DateTime, 168  
MML, 152  
Mod (modulo) key, 37  
Month  
    DateTime, 168  
MonthName  
    DateTime, 168  
MoveTo  
    Graphics, 207  
music (PLAY), 152  
Music Macro Language (MML),  
    152  
N (count), 61  
n, Text, 210  
N<sub>a</sub> (Avogadro's number) key,  
    45  
Name, Data field, 163  
NaN key, 45  
NEXT, 143  
NOT, 118  
NOTTI, 269  
oct key, 51  
Octal numbers, 51  
Opacity, Graphics, 202  
OR, 119  
Or (|) key, 54  
Output screen, 105  
close, 106  
CLS, 134  
larger, 106  
PRINT, 154  
smaller, 106  
P10 (10% percentile), 64  
P90 (90% percentile), 63  
package  
    about, 95  
    delete, 96  
    how to add, 317  
    how to display properties, 92  
    how to export, 96  
    how to import, 93  
    made up of programs, 97  
Parentheses (Calculator), 25  
Parse  
    DateTime, 166  
    String, 222  
PAUSE, 152  
Percent (%) key (%), 32  
Percent Discount, 32  
Percent Formatting (Calculator),  
    44  
PI, 115  
PickDevicesName, 238  
Picklocation  
    Data, 164  
pitch (BEEP), 133  
PLAY, 152  
Polygon, 207  
Population Standard Deviation,  
    61  
Pos  
    String, 224  
Post  
    Http, 179  
pressure, Infineon, 261  
Pretty-print, 130

PRINT, 154  
program  
    about, 99  
    add, 97  
    delete, 100  
    edit, 98  
    edit dialog, 102  
    edit from the About dialog,  
        100  
    how to add, 318  
    how to bind to a program  
        key, 321  
    how to run, 320  
    properties, 98  
    run, 98  
    running from edit dialog, 102  
    saving while editing, 102  
program list  
    how to display, 92  
Programmer's calculator, 51  
Puck.Js, 272  
Puck.mag, 273  
Put  
    Http, 179  
p-value, 66  
Q1 (1<sup>st</sup> quartile), 63  
Q2 (2<sup>nd</sup> quartile), 63  
Q3 (3<sup>rd</sup> quartile), 63  
r→d key, 33  
radians and degrees  
    (Calculator), 33  
radians key, 33  
RAND (set random seed), 157  
Random  
    RemoveAlgorithm, 141  
Random Numbers (BASIC), 157  
Random Numbers (Calculator),  
    41  
READ, 158  
Read, Ardudroid, 249  
Read, Bluetooth device, 241  
ReadAll  
    File, 171  
ReadCachedByte, 241  
ReadCachedBytes, 242  
ReadLines  
    File, 171  
ReadPicker  
    File, 171  
ReadRawByte, 241  
ReadRawBytes, 242  
Recall (M→), 28  
Rectangle  
    Graphics, 205  
Regression, 65  
Relative Standard Deviation, 61  
REM (comment), 159  
RemoveAlgorithm  
    array, 141  
Replace  
    String, 224  
RequestActive, 191  
RequestRelease, 191  
Reservoir Sampling  
    array, 141  
RESTORE, 158  
Rfc1123  
    DateTime, 168  
Rfcomm, 248  
Rfcomm, Bluetooth, 235  
RIGHT, 123  
RND, 115, 157  
rnd key, 41  
rnd N key, 41  
Robust Statistics, 63  
rotate left (<<+○) key, 56  
rotate right (>>+○) key, 56  
Rotate, Graphics, 202  
Round key, 39  
Rounding (Calculator), 39

RSD, 61  
RTF, Pretty-print, 130  
Ruuvitag, 237  
Sales Tax, 32  
Sample Standard Deviation, 61  
save (export) package, 96  
SaveScreenToMemory, 257  
Scaling  
    Graphics, 210  
Scatterplots, 68  
Scientific (exponential) notation  
    (BASIC), 110  
Scientific (exponential)  
    Notation (Calculator), 31  
screen clear key (BASIC), 97  
Screen Extension, 191  
Screen.ClearLine, 191  
Screen.ClearLines, 191  
Screen.H, 192  
Screen.RequestActive, 191  
Screen.RequestRelease, 191  
Screen.W, 192  
Second  
    DateTime, 168  
selector, Gopher, 176  
Sensor Extension, 212  
Sensor.Camera, 212  
Sensor.Location, 216  
Sensor.Microphone, 217  
SensorTag 2541 (original), 284,  
    287  
SensorTag\*, 284, 287  
Serial Port Protocol, 248  
Serial-port, Bluetooth, 235  
server, Gopher, 174  
service, Bluetooth, 240  
ServoAttach, Ardudroid, 250  
ServoWrite, 250  
Set  
    DateTime, 167  
SetAlarmTime, 270  
SetColor, beLight, 254  
SetColor, MagicLight, 263  
SetColor, NOTTI, 267, 270  
SetColorCustom, NOTTI, 270  
SetColumn, 257  
SetInterval, System, 226  
SetMoved, Graphics, 198  
SetName, 257  
SetName, NOTTI, 270  
SetNameArbitrary, 257  
SetNameArbitrary, NOTTI, 270  
SetOff, MagicLight, 263  
SetOn, MagicLight, 263  
SetPanel, 257  
SetPixel, 257  
SetPoints, Polygon, 207  
SetPosition  
    Graphics, 198  
SetPressed, Graphics, 198  
SetProperty, array, 140  
SetReleased, Graphics, 198  
SetRow, 257  
SetScale  
    Graphics, 210  
SetScaleWindow  
    Graphics, 210  
SetSize  
    Graphics, 198  
SGN (BASIC), 123  
shift left (<<) key, 56  
shift right (>>) key, 56  
SILENT, 160  
SIN (BASIC), 122  
Sin key, 33  
Size  
    File, 170, 173  
Size, Screen.GX and GY, 193  
Skoobot, 277  
Slant Robotics, 282

Slider, 208  
Slope, 65  
Smart quotes, 131  
  and strings, 111  
SPC, 126  
SPEAK [VOICE <voice>] <text>, 159  
specializations, Bluetooth, 246  
SPP, 248  
SPP, Bluetooth, 235  
SQR (BASIC), 122  
Standard Deviation, 61  
Stars, how to draw, 208  
Start  
  Gopher, 177  
statement, 126  
statement terminator (not in BASIC), 109  
Statistical calculator, 58  
Statistics  
  Entering data, 59  
StdErr Line, 65  
StdErr Slope, 65  
STOP, 160  
STOP (FOREVER), 146  
STOP (PLAY), 154  
Store ( $\rightarrow$ M) key, 28  
String constants, 111  
String Extension, 220  
Student's t-test, 66  
subscript letters in variable  
  names, 112  
Subtract  
  Dates, 50  
  DateTime, 167  
Sum, 61  
SumOfSquares, 141  
SWAB (swap bytes) key, 55  
SyncTime, 258  
SyncTime, NOTTI, 270  
System Extension, 226  
t statistic, 66  
TAN (BASIC), 122  
Tan key, 33  
TARE, Altimeter program, 305  
temperature, Infineon, 262  
Tests, Running BASIC, 107  
Texas Instruments, 284, 287  
Texas Instruments, beLight  
  CC2540T, 254  
Text, Screen.Graphics, 210  
TI Display (Watch), 293  
TI SensorTag 2650, 292  
Time  
  DateTime, 168  
TimeHHmm  
  DateTime, 168  
ToLower  
  String, 224  
ToUpper  
  String, 224  
Trace, System, 227  
trigonometry, 33  
T-Tests, 66  
Tukey boxplots, 58  
Unicode minus signs, 111  
Unicode table (BASIC), 131  
Unicode table (Calculator), 74  
un-indent lines, 104  
Update  
  Graphics, 200  
UseScale  
  Graphics, 211  
VAL, 126  
Variables, 112  
  ending in \$ (dollar sign), 112  
Version, System, 227  
Visually Perfect Algebraic  
  Method, 26  
WAIT (FOREVER), 146

- WAIT (PLAY), 153
- Walteros, 282
- Watch, 293
- Watch, "Bluetooth", 236
- Watch, "Eddystone", 236
- Watch, "Eddystone-URL", 237
- Watch, "RuuviTag", 237
- Watch, Bluetooth, 236
- WatchPrint, 293
- Welch's t-test, 66
- Witti Design, 256, 269
- word (Π) key, 55
- Word output, pretty-print, 130
- Write, Ardudroid, 250
- WriteCallbackDescriptor,
  - Bluetooth, 243
- WriteClientCharacteristicConfig
  - urationDescriptorAsync, 243
- WriteLine
  - File, 173
- WritePicker
  - File, 173
- WriteText
  - File, 173
- Ȑ (mean), 61
- X1, Graphics, 202
- x<sup>2</sup> (Square) key, 30
- X2, Graphics, 202
- x<sup>3</sup> key, 37
- Xor (^) key, 54
- x<sup>y</sup> key, 38
- XY Scatterplots, 68
- yv/x key, 38
- Y1, Graphics, 202
- Y2, Graphics, 202
- YAxisMax
  - GraphXY, 196
- YAxisMin
  - GraphXY, 196
- Year
  - DateTime, 168
- Zira (SPEAK VOICE), 159
- π key, 45
- s (sample standard deviation),
  - 61
- Σ (sum), 61
- σ<sub>n</sub> (population standard deviation), 61