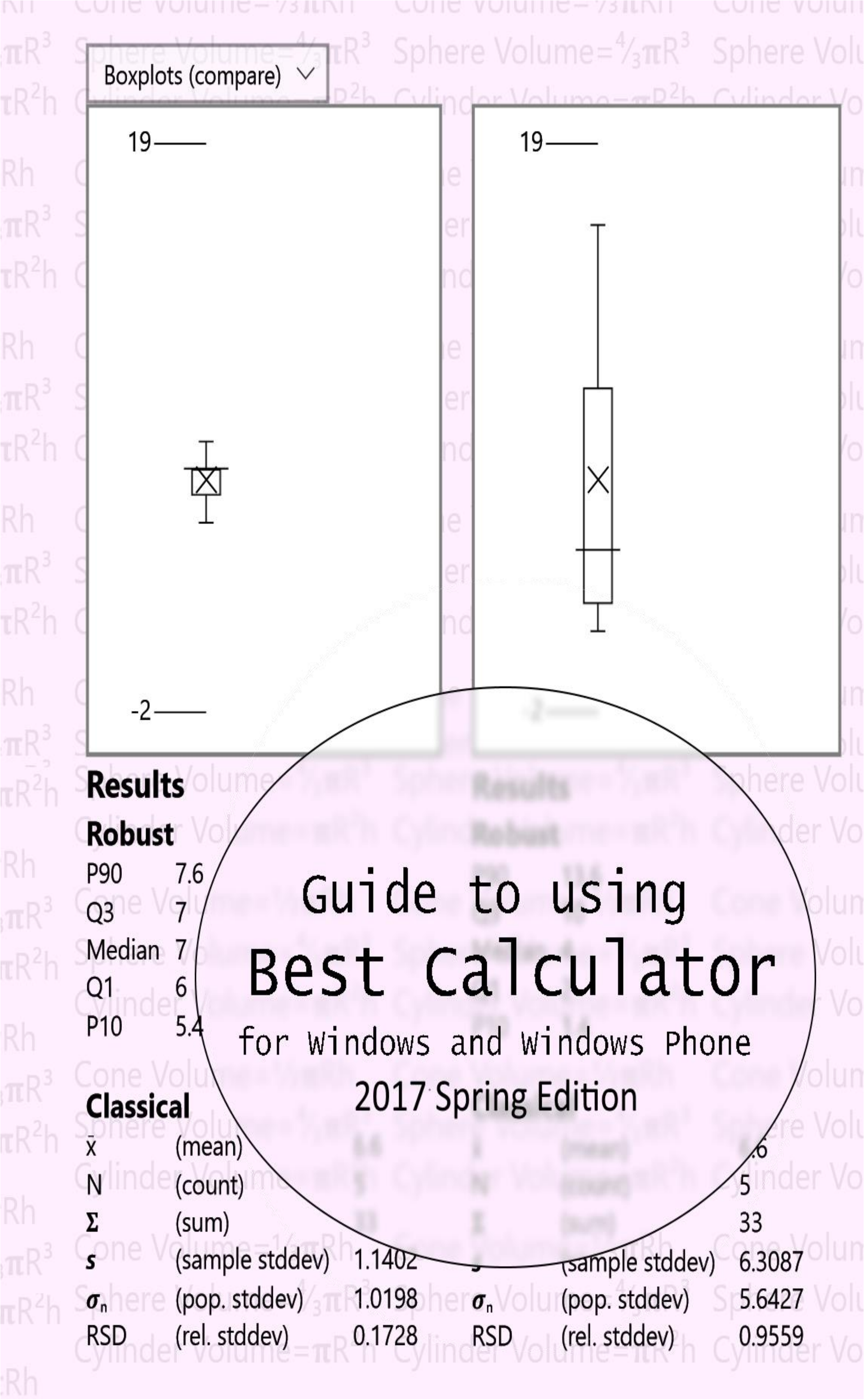


Results

Robust

P90	7.6
Q3	7
Median	7
Q1	6
P10	5.4

Classical

x̄	(mean)				.6
N	(count)				5
Σ	(sum)				33
s	(sample stddev)	1.1402		(sample stddev)	6.3087
*σ*ₙ	(pop. stddev)	1.0198	*σ*ₙ	(pop. stddev)	5.6427
RSD	(rel. stddev)	0.1728	RSD	(rel. stddev)	0.9559

Guide to using Best Calculator

for Windows and Windows Phone

2017 Spring Edition

BC BASIC Reference manual and tutorial

including

The Best Calculator Reference Manual

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/9wzdncrdfd6x

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

TABLE OF CONTENTS

1 A Quick Tour of Best Calculator 19

2 Numbers and Common Calculations 21

2.1 Simple Arithmetic 21

2.2 Chain calculations 21

2.3 Algebraic Entry and Parentheses 21

2.4 Editing errors and clearing the display 22

3 Memory Keys 23

3.1 Memory operation 23

3.2 Example 23

3.3 Memory Add and Subtract 24

4 More Math 25

4.1 Square (x^2) key 25

4.2 Square Root (√) key 25

4.3 Inverse (1/x) key 25

4.4 Change Sign (±) key 25

5 Scientific Notation 26

5.1 Using the EE key 26

6 Percent Key % 27

6.1 Percent (%) key 27

6.2 Calculating sales tax 27

6.3 Calculating sales with a percent discount 27

7 Trigonometry keys on the Advanced calculator 28

7.1 Calculate in degrees or radians 28

7.2 Sin, Cos, Tan 28

7.3 Inverse Sin, Cos, Tan 29

8 Logarithms on the Advanced calculator 30

9 Powers, roots, x!, mod on the Advanced calculator 31

9.1 Factorial (x!) key 31

9.2 Mod Key 31

9.3 Cube (x^3) and Cube Root ($\sqrt[3]{x}$) keys 31

9.4 Arbitrary Power (x^y) and Root (y√x) keys 32

10 Rounding and abs on the Advanced calculator 33

11 Random Numbers on the Advanced calculator 34

12 Formatting 35

12.1 Standard formatting 35

12.2 Exponent formatting 36

12.3 Fixed formatting 36

12.4 Natural formatting 36

12.5 Percent formatting 37

13 Constants 38

13.1 π and e 38

13.2 g_n c and N_a 38

13.3 NaN, ∞ and -∞ 38

14 Memory Page 39

14.1 Show the operation of Memory0 39

14.2 Saving and Roaming 39

14.3 Name a memory cell 39

14.4 Get from display and Copy to display 40

14.5 Clear 40

14.6 Memory and BC Basic 40

15 Dates Page 41

15.1 Compare Dates 41

15.2 Convert to Gregorian 41

15.3 Add days 42

16 Hex, Decimal, Octal, Binary on the Programmer’s calculator 43

16.1 Setting the mode 43

16.2 Converting between bases 43

17 Bit Operators on the Programmer’s calculator 44

17.1 B# (count bits) 44

17.2 Inverse (~) and 2’s complement 44

17.3 And (&), Or (|) , Xor (^) 44

18 Bytes and Swabbing on the Programmer’s calculator 45

18.1 Bytes 45

18.2 🔄SWAB (Swap Bytes) key 45

19 Shift operators on the Programmer’s calculator 46

20 Programmer’s Math 47

21 Statistical calculator 48

21.1 Parts of the Statistical calculator screen 48

21.2 Entering Data 49

21.3 Classical Statistics (Results) 50

Robust Statistics (Results) 51

Regression (Results) 52

21.4 Compare with T-Tests (Results) 53

21.5 Boxplots (Graph) 54

21.6 XY Scatter-plots (Graph) 55

22 Conversions 56

22.1 Converting 56

22.2 Copy to and from the Calculator results display 56

22.3 Area conversions 57

22.4 Energy Conversions 57

22.5 Length Conversions 57

22.6 Temperature Conversions 58

22.7 Weight 58

22.8 Farm Volumes (US) 59

23 Ascii Table 60

24 Unicode 61

24.1 Search Rules 61

24.2 Copying characters 61

25 Advanced Windows Features 62

25.1 Shortcut on the desktop 62

25.2 Set as the Calculator Key 63

26 A brief historical note 64

27 Equation Input: your first program 65

27.1 What is a program? 65

27.2 How do I run my program? 65

27.3 How can I print some text? 65

27.4 How can my program use the calculator value? 66

27.5 How can I write an equation? 66

27.6 How can I write to the calculator? 66

27.7 Why do some examples start with an = sign? 67

27.8 How can I make several different programs? 67

27.9 What are all the keys on the screen? 67

27.10 How can I learn more? 68

28 Sigma Function: advanced programming 69

29 What all can you do in the BC BASIC environment? 71

29.1 All the main edit dialogs 71

29.2 Library of Packages 72

29.3 Library Properties (Import BC BASIC package) 74

29.4 Bind a program to a key 75

29.5 About this package 76

29.6 List of programs 78

29.7 About this program 80

29.8 Edit Program 82

29.9 Output screen 84

30 BASIC Language Reference 86

30.1 Program structure 86

30.2 Flags 87

30.3 Numbers and Strings and Variables 87

30.3.1 Numbers 87

30.3.2 Strings....88

30.3.3 Variables, GLOBAL, and DIM....88

30.4 <expression> overview....90

30.4.1 Quick introduction to expressions....90

30.4.2 Expression Rules....90

30.4.3 negate, power, root prefix and postfix operators....92

30.4.4 Operators + - * / and more....92

30.4.5 The INPUT expression....96

30.5 Math Functions....96

30.5.1 Trigonometry functions SIN COS TAN ASN ACS ATN....97

30.5.2 Logarithm and Power functions LN EXP SQR....97

30.5.3 Rounding and sign functions SGN ABS INT....97

30.6 String functions (LEFT, MID, RIGHT, LEN, CHR$, CODE, VAL)....98

30.6.1 LEFT (string, count)., MID (string, index, count) and RIGHT (string, count) 98

30.6.2 LEN string....99

30.6.3 CHR, CODE....99

30.6.4 VAL(string)....100

30.7 <statement> overview....101

30.8 Packages and Programs....102

30.8.1 Inside a package file....103

30.9 Picking a package for your program....103

30.9.1 Creating common functions for several programs....103

30.9.2 Exporting packages....103

30.10 All of the Special Symbols....104

31 BASIC Statements Reference....105

31.1 [CALL] <function> (<expression>, …)....105

31.2 CLS [<color>, <color>] and PAPER <color>....106

31.3 CONSOLE <expression> [, <expression>] 107
31.4 DIM <name> ([size]) and array methods 108
31.5 DIM'd array methods and properties 110
31.5.1 Count property 110
31.5.2 Max and Min properties 110
31.5.3 MaxOf(column) and MinOf(column) properties 110
31.5.4 Add() method and MaxCount and RemoveAlgorithm properties 110
31.5.5 AddRow() method 111
31.6 DUMP 111
31.7 FOR <variable> = <start> TO <end> [STEP <step>] … NEXT <variable> 112
31.8 FUNCTION <name> (<args>) … RETURN [<value>] 115
31.9 GLOBAL <variable> 115
31.10 GOSUB <linenumber> and RETURN 116
31.11 GOTO <linenumber> 116
31.12 IF (<expression>) THEN <statement> [ELSE <statement>] 117
31.13 IF (<expression>) … [ELSE …] ENDIF 117
31.14 IMPORT FUNCTIONS FROM "program" 118
31.15 INPUT <variable> 119
31.16 (LET) <variable> = <expression> 119
31.17 PAUSE <frames> 120
31.18 PRINT [AT row,col] <expression> [(, or ;) [AT row,col] <expression>]* 121
31.19 RAND <seed> & the RND value 122
31.20 REM comment words to the end of the line 123
31.21 STOP and END 123
32 Extensions Reference 125
32.1 Calculator.Value and Calculator.Message Extension 125
32.2 DateTime Extension 126
32.2.1 DateTime.GetNow() 126
32.2.2 DateTime.Subtract (datetime) 126

32.2.3 Hour, Minute, Second, Year, Month, Day and DayOfWeek properties....126

32.2.4 DateTime.Date, DateTime.Time and DateTime.TimeHHmm.....126

32.2.5 DateTime.Iso8601 and DateTime.Rfc1123 127

32.2.6 DateTime.AsTotalSeconds 127

32.2.7 Example of all the properties....127

32.3 File Extension.... 128

32.3.1 file=File.AppendPicker(), file.AppendLine, file.AppendText() and file.Size 128

32.3.2 file=File.ReadPicker(".txt") , file.ReadAll(), file.ReadLines().......129

32.3.3 file=File.WritePicker(), file.WriteLine, file.WriteText() and file.Size 130

32.4 Http Extension 131

32.4.1 Http.Get(url, [headers]) 131

32.4.2 Http.Post() and Http.Put()....132

32.5 Math Extension 133

32.5.1 Trigonometry (Math.Sin (radians) and more)....133

32.5.2 Rounding and sign (Floor(), Round() and more) 134

32.5.3 Logarithm and power functions (Math.Log, Math.Exp, and more) 137

32.5.4 Math.Factorial and Math.IsNaN....138

32.5.5 Math.PI, Math.E and Math.NaN values 138

32.6 Memory Extension 139

32.6.1 Memory[<expression>] and Memory.<name>....139

32.6.2 GetOrDefault and IsSet functions 140

32.6.3 Memory technical details....141

32.7 Screen Extension 143

32.7.1 Screen.ClearLine(<line>) and Screen.ClearLines(<from>, <to>) 143

32.7.2 Screen.RequestActive() and Screen.RequestRelease() 143
32.7.3 Screen.H and Screen.W Extension 144
32.8 Screen.Graphics() Extension 145
32.8.1 Circle(X, Y, radius) Line(X1, Y1, X2, Y2) Rect(X1, Y1, X2, Y2) 145
32.8.2 SetPosition(X,Y) and SetSize(H, W) 146
32.8.3 GraphXY(data) 147
32.8.4 GraphY(data) 147
32.8.5 Updating Data with graph.Update() and PAUSE 148
32.9 String Extension 150
32.9.1 String.Escape ("csv", <string or array>) 150
32.9.2 String.Escape ("json", <string or array>) 152
32.9.3 String.Parse("csv", <data string>) 152
32.9.4 String.Parse ("json", <data string>) 153
33 Bluetooth Programming with Best Calculator, IOT edition 155
33.1 Programming Bluetooth using BC BASIC 155
33.2 Initializing your device and available properties 156
33.2.1 Error handling and Bluetooth 158
33.3 The objects you use when programming your Bluetooth device 159
33.3.1 The Bluetooth object 159
33.3.2 The Bluetooth.Devices object (Array / ObjectValueList) 160
33.3.3 Individual Bluetooth devices from Bluetooth.Devices 160
33.3.4 Specializations 161
33.4 Selecting a device with PickDevicesNames and more 161
33.4.1 Bluetooth.PickDevicesName(<name pattern>) 161
33.4.2 Bluetooth.DevicesName(<name pattern>) 163
33.4.3 Bluetooth.Devices() 163
33.5 Reading data from raw Bluetooth services and characteristics 163
33.5.1 Direct device Read routines 164

33.6 Using callbacks to read data 166

33.7 Using the specializations for specific devices 169

34 Bluetooth Specializations for specific devices 170

34.1 BBC micro:bit 171

34.2 beLight CC2540T Light development kit 174

34.3 DOTTI device 175

34.4 Hexiwear wearable platform 178

34.5 MagicLight and Flux light 180

34.6 MetaWear MetaMotion R device 181

34.7 NOTTI device 186

34.8 TI SensorTag 2541 (original version) 189

34.9 TI SensorTag 1350 (2016 version) 192

35 Complete Examples 197

35.1 Connecting to Microsoft Flow 197

35.1.1 Set up the flow at Microsoft Flow 197

35.1.2 Write the BC BASIC program 199

35.2 Hiking with an altimeter 202

35.2.1 The Graph program 202

35.2.2 The Altitude program 206

35.2.3 The TARE program 207

36 Graphics and BC BASIC 209

36.1 Fixed character screen commands 210

36.2 Console Commands 212

37 Using the library, step by step 212

37.1 Add a new package for your program 214

37.2 Add a new program to your package 215

37.3 Edit the program to do the conversion 215

37.4 Run the program to test it 217

37.5 Bind the program to a key 218

37.6 Next steps 219

38 Appendix: Release Notes 220

39 Appendix: Sample programs 221

39.1 BT: An Overview of Bluetooth 221

39.1.1 List Bluetooth devices 221

39.1.2 Pick a Bluetooth device 222

39.1.3 Power 222

39.2 BT: BBC Microbit....................223

39.2.1 Accelerometer....................223

39.2.2 Button 224

39.2.3 Compass 225

39.2.4 Magnetometer 226

39.2.5 SetLed....................227

39.2.6 Status....................228

39.2.7 Temperature 228

39.2.8 Write (text, speed) 230

39.3 BT: beLight....................230

39.3.1 Green....................230

39.3.2 Red 231

39.4 BT: Dotti....................232

39.4.1 An introduction 232

39.4.2 Change Mode 232

39.4.3 List BT Devices....................233

39.4.4 Load screen from memory....................234

39.4.5 Raw Bluetooth commands....................235

39.4.6 SetAnimationSpeed....................236

39.4.7 SetColumn and SetRow to random lines 237

39.4.8 SetName of a Dotti device 238

39.4.9 SetPixel to random green dots....238

39.4.10 SetPixel to write a single green dot 239

39.4.11 Sync Time 239

39.5 BT: Hexiwear.... 240

39.5.1 Accelerometer....240

39.5.2 Compass 241

39.5.3 List Information....242

39.5.4 Raw Access to Hexiwear 243

39.5.5 Read All 244

39.5.6 Set notification count....246

39.5.7 SetTime 247

39.6 BT: MagicLight 247

39.6.1 Green....247

39.6.2 Off 248

39.6.3 On....248

39.6.4 Red 249

39.7 BT: MetaWear MetaMotion 249

39.7.1 _Basics....250

39.7.2 Accelerometer....250

39.7.3 Altimeter 252

39.7.4 Ambient Light Sensor....253

39.7.5 Barometer 254

39.7.6 Black 256

39.7.7 Blue Pulse....256

39.7.8 Buttons....257

39.7.9 Green....258

39.7.10 Gyroscope 259

39.7.11 LED Off....260

39.7.12 LED On....260

39.7.13 Magnetometer....261

39.7.14 Save Data To File....262

39.7.15 Temperature....265

39.8 BT: Notti....266

39.8.1 An introduction....266

39.8.2 Change Mode....267

39.8.3 NOTTI timer....267

39.8.4 Raw Bluetooth commands....269

39.8.5 Set Alarm....269

39.8.6 SetColor to change the NOTTI color to blue....270

39.8.7 SetColor to change the NOTTI color to green....271

39.8.8 SetColorCustom animates the color from red to blue and back again 271

39.8.9 SetName of a NOTTI device....272

39.8.10 Sync Time....272

39.9 BT: SensorTag 1350....273

39.9.1 Accelerometer Gyroscope and Magnetometer....273

39.9.2 Accelerometer Off....275

39.9.3 Barometer....275

39.9.4 Button....277

39.9.5 Humidity....277

39.9.6 IO....278

39.9.7 IR....279

39.9.8 Optical Sensor....280

39.10 BT: SensorTag 2541....281

39.10.1 Accelerometer....281

39.10.2 Accelerometer Off....282

39.10.3 Accelerometer to Magic Light....283

39.10.4 Barometer....284

39.10.5 Button....285

39.10.6 Gyroscope....286

39.10.7 Humidity....287

39.10.8 IR....288

39.10.9 Magnetometer....290

39.10.10 Raw access to SensorTag....291

39.10.11 Raw callback with the SensorTag....292

39.11 EX: BC BASIC Quick Samples....293

39.11.1 Colorful Countdown....293

39.11.2 Grams of Fat to Calories....295

39.11.3 Miles per Gallon....295

39.11.4 Right Triangle calculator....296

39.11.5 Tip Calculator....297

39.11.6 Welcome to BC BASIC....298

39.12 EX: Files, CSV and JSON, HTML, Flow....299

39.12.1 Appending to a file....299

39.12.2 Http.Get(url, headers) reads data from the internet....300

39.12.3 Microsoft Flow example....301

39.12.4 Read Entire File....304

39.12.5 Reading a CSV file....305

39.12.6 Writing to a file (including CSV)....305

39.13 EX: Financial....306

39.13.1 Common Tip Values....306

39.13.2 Compound Interest 308

39.13.3 Doubling Time 309

39.13.4 Future Value 310

39.13.5 Money Conversion 311

39.13.6 Present Value 312

39.13.7 Return on Investment 313

39.14 EX: Real Estate 314

39.14.1 Acres to square feet 314

39.14.2 Debt to Income calculations 314

39.14.3 Minimum and Maximum density 315

39.14.4 Rectangle in feet to acres 316

39.14.5 Square feet to acres 317

39.15 EX: Space and Astronomy 317

39.15.1 Arc Length 317

39.15.2 AU to Meters 318

39.15.3 Conversion Library 318

39.15.4 Distance to horizon 320

39.15.5 Lightyears to Parsecs 321

39.15.6 Meters to AU 321

39.15.7 Parsecs to Lightyears 321

39.15.8 Rocket Equation 322

39.16 EX: Statistics 324

39.16.1 Finite Population Correction 324

39.16.2 Margin of Error 324

39.16.3 Pfail 325

39.16.4 Sample Size 325

39.16.5 Sample Size Library 327

Index to Best Calculator Manual 329

Best Calculator editions

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

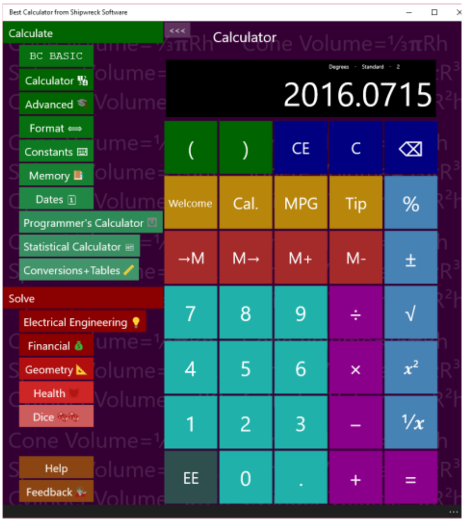
The Best Calculator, IOT edition takes the familiar and powerful Best Calculator and adds the ability to connect to a range of IOT (Internet of Thing) devices. All the parts in common with the standard edition of Best Calculator are free; the rest require a small payment.

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. There is a free trial so you can make sure that the app has the capabilities you need.

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 paid edition.

1 A Quick Tour of Best Calculator



Use the menu on the left to pick a calculator page to use

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.

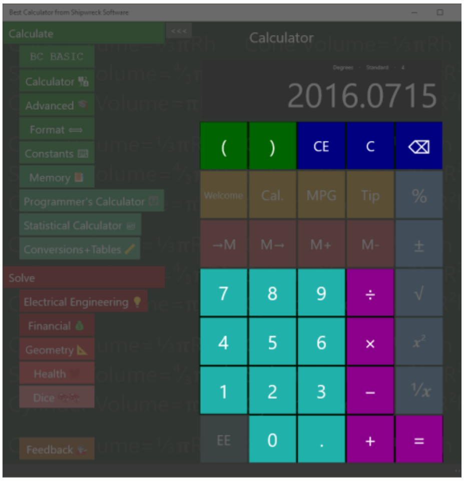
2 Numbers and Common Calculations

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 × 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 (× 20)

```
Key in: 15 × 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 × 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+2×3 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 ⌫ keys are used to edit input errors and clear the calculator. The Clear Al 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’.

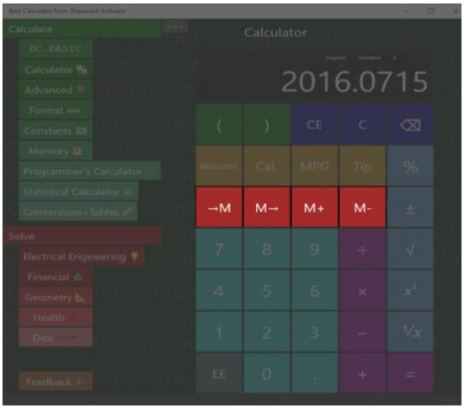
```
Key in: 22 + 33 ⌫ 7 =
Result:   59
```

The ⌫ (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.



Basic Memory operations include get from memory, store into memory

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
```

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.

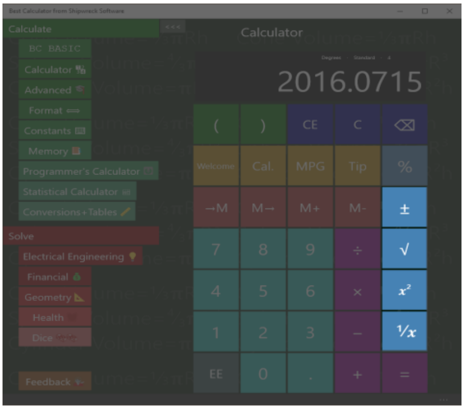
4 More Math

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 x²
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 × 5 =



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

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

4.2 Square Root (√) Key

The √ 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 √
Answer: 5
```

4.3 Inverse (*1/x*) Key

What's ⅓ + ⅓?

```
Key in: 3 1/x + 3 1/x =
Answer: 0.666
```

4.4 Change Sign (±) Key

The ± 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 6.022×10^{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 3.34×10^{22}

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

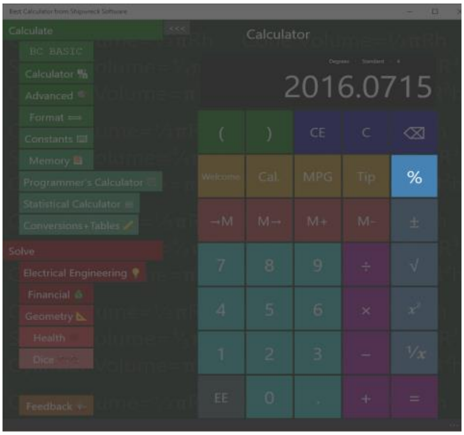
The mass of a single oxygen atom is about 2.68×10^{-26} kilograms. What is the mass of 20 atoms of oxygen?

Key in: 2.68 EE – 26 × 20
Answer: 5.36E-25 or 3.34×10^{-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



The percent key is used for taxes and more

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)

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

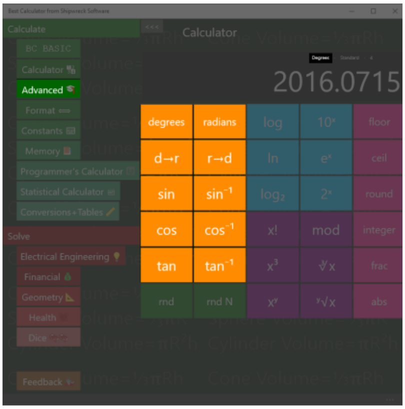
You are buying an item with a 20% pre-tax discount, and the sales tax is 5%. What's the total?

```
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° 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°

```
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

Calculate the cosine of ¼π

```
Key in: radians 0.25 × π = cos
Answer: 0.7071
```

You can perform the same calculation using parenthesis

```
Key in: radians ( 0.25 × π) 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° and 360° degrees or 0 to 2π 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.

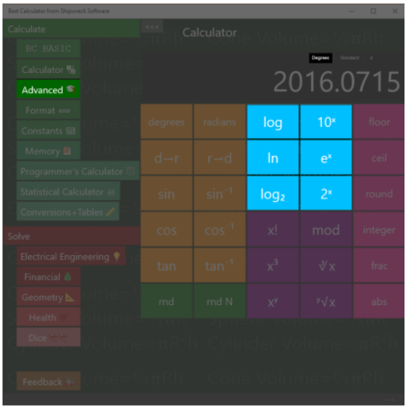
The notation sin^{-1} is the John Herschel notation; it means “inverse sine” and not “raised to a power”.

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$_2$ key calculates using base 2 (binary, also called lb)

Each key is paired with the corresponding power key: 10^x, e^x, and 2^x.

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
```

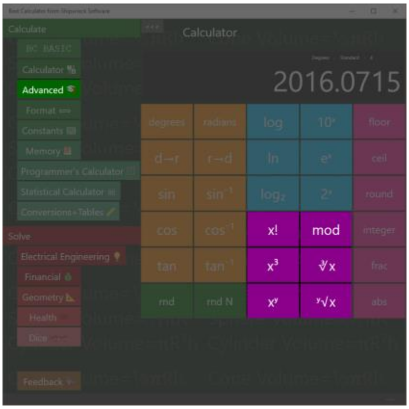
9 POWERS, ROOTS, X!, MOD ON THE ADVANCED CALCULATOR

9.1 FACTORIAL (X!) KEY

Calculate 6! (6 factorial)

```
Key in: 6 x!
Answer: 720
```

6! is another way of writing 6 x 5 x 4 x 3 x 2 x 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 reminder 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^{rd} power, or 4.5 × 4.5 × 4.5)

```
Key in: 4.5 x³
Answer: 91.125
```

What is cube root of 27?

```
Key in: 27 ∛
Answer:  3
```

9.4 Arbitrary Power (X^Y) and Root (Y√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^y 4 =

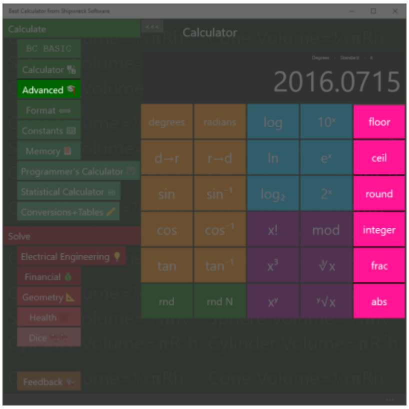
Answer: 1296

What is the 4th root of 32?

Key in 32 y√x 4

Answer: 2.3784

10 Rounding and ABS on the Advanced Calculator



Rounding, remainder and absolute value

Rounding keys are in the advanced calculator.

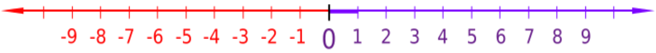
Floor: round downwards to be a smaller number. The floor of a negative number (like -3.5) is rounded to a smaller number (-4)

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.

(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

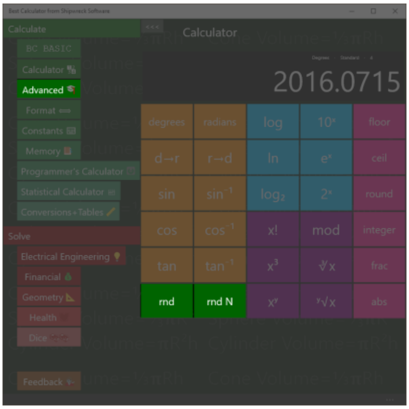
```
Key in: 5.83 remain
Answer: 0.83
```

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



Random Numbers

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.

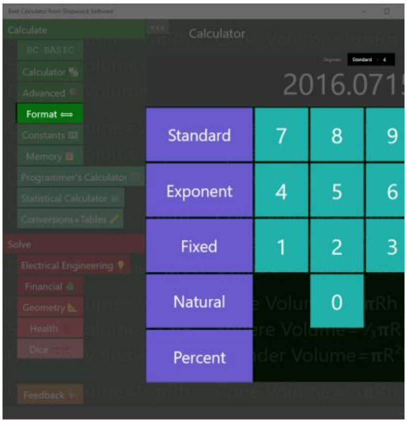
Make a random number between 1 and 12

```
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)
```

12 Formatting



Change how your results are displayed with the Format page.

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)

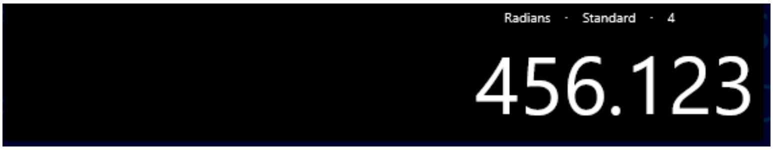
You can pick how many numbers after the decimal place to display.

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”

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).


Radians · Standard · 4

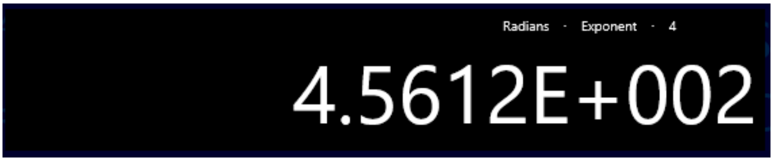
456.123

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.


Radians · Exponent · 4

4.5612E+002

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

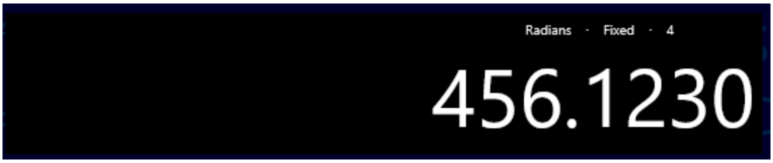
A negative exponent (4.5612E-002) is less than zero (0.045612)

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


Radians · Fixed · 4

456.1230

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

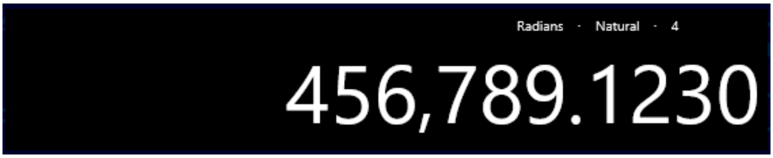
If the number is too large to fit into the display, the display will be resized. After a certain point, the number will no longer fit, and will be truncated.

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 comas separating the units.



Radians · Natural · 4

456,789.1230

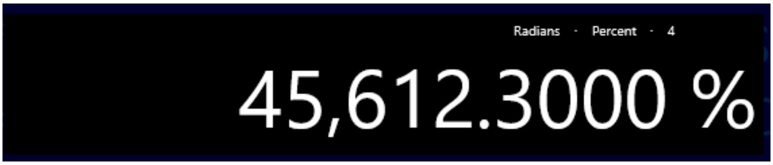
If the number is too large to fit into the display, the display will be resized. After a certain point, the number will no longer fit, and will be truncated.

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

12.5 PERCENT FORMATTING

Regular number or exponent?	Regular number
Special features	Number is display 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.



Radians · Percent · 4

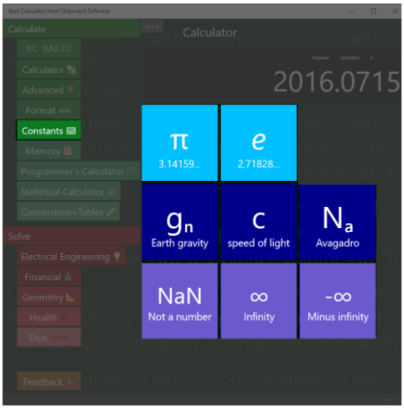
45,612.3000 %

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 45**7**.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.



Best Calculator includes useful constants

13.1 π and e

Press the π key to set the display to the value of π (about 3.1416).

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

Calculate 4 * π

```
Key in: 4 × π =
Answer: 12.5664
```

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^2.

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

N_a is Avogadro's number (about 6.022x10^{23}) is the number of atoms or molecules in one *mole* of a substance. For example, there are N_a atoms of carbon (specifically, ^{12}C) in 12 grams of carbon.

13.3 NaN, ∞ and -∞

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

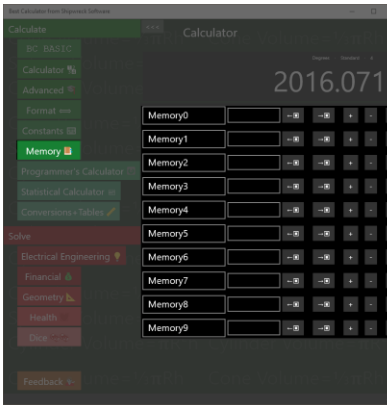
NaN (Not a Number) means that a numeric value is not meaningful. For example, an inverse sin (sin^{-1}) is only meaningful for input values -1 to 1. Calculating sin^{-1} of 2 will result in a NaN result.

∞ is positive infinity, and -∞ 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.



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

14.1 SHOW THE OPERATION OF MEMORY0

The main calculator screen includes 4 memory keys (→M, M→, 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!

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 ←▣ key gets data from the display into the memory slot.

The →▣ 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.

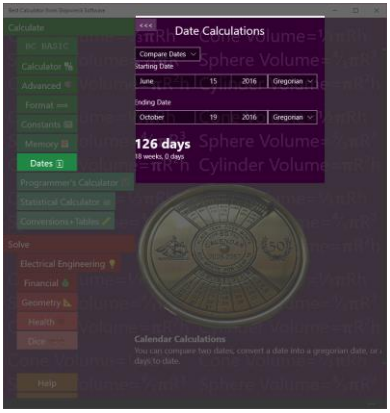
15 Dates Page

The Dates page lets you perform various date calculations.

15.1 Compare Dates

How many days are there between June 15th and October 19th?

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

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



Date Calculations

Compare Dates
Starting Date
June 15 2016 Gregorian
Ending Date
October 19 2016 Gregorian

126 days
18 weeks, 0 days

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 (הַלּוּחַ הָעִבְרִי) is a lunisolar Jewish calendar
- Hijiri (گاه‌شماری 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 6th 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 6th, 2016 and set the calendar type to Julian.

Answer: January 19, 2016

15.3 ADD DAYS

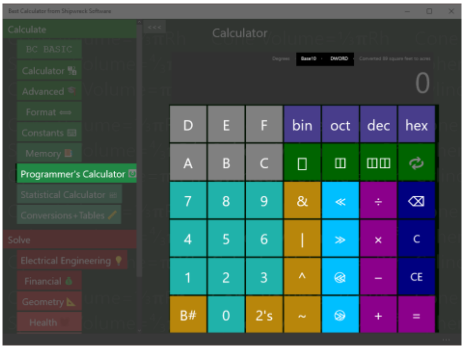
A project is started on April 19, 2016, and will take 21 days. What day will the project end on?

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

Answer: Tuesday, May 10, 2016

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.



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

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

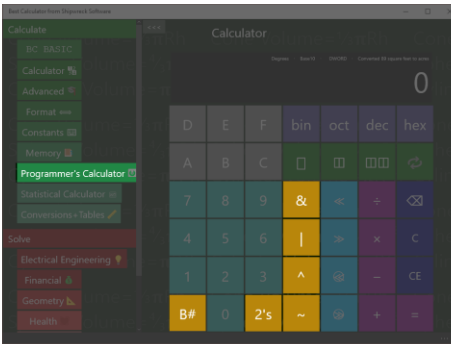
To convert from the current base to a different base, simply press the new base number. The existing display will be converted to the new base

Convert $2C_{16}$ into decimal

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

17 Bit Operators on the Programmer's Calculator

The Bit operators are part of the Programmer's calculator



Basic Memory operations: get from memory, store into memory

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.

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
```

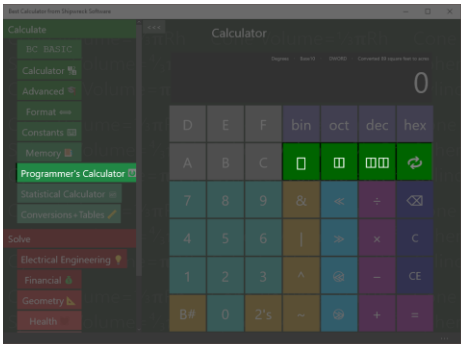
17.3 And (&), Or (|) , Xor (^)

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



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

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.

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:

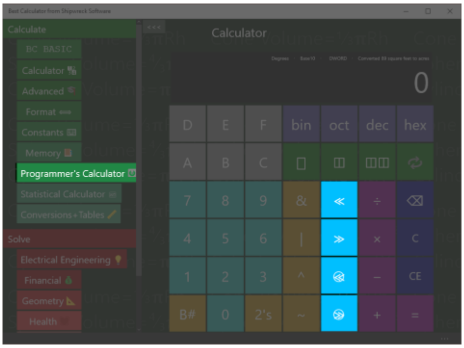
- ◫ switches to WORD (2 byte) mode
- dec switches to decimal mode
- 8 0 puts the number 80 into the display
- 🔄 (SWAB) switches the first and second byte around

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.

≪ is shift left
≫ is shift right
≪+○ is rotate left
≫+○ is rotate right



Shift operators: left and right, left and right rotate (ring shifts), and the byte/word/dword display

Shift hex F0 left by 2 bits

```
Key in: hex F 0 ≪ 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.

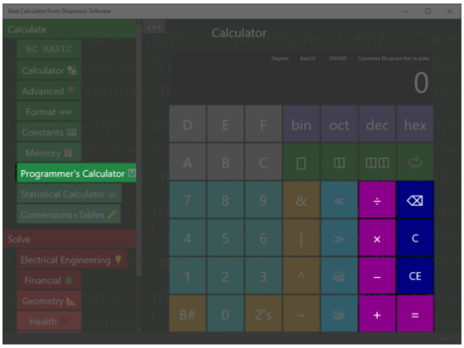
Ring-shift left F0 by 1 bit as a single byte

```
Key in:  ▯ hex F 0 ≪+○ 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 performs standard math operations

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

Divide 7 by 3

Key in: 7 ÷ 3 =
Answer: 2

Unlike the regular calculator, where 7 ÷ 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 × 99 in byte mode

Key in: ▯ 9 9 × 9 9 =
Answer: 73

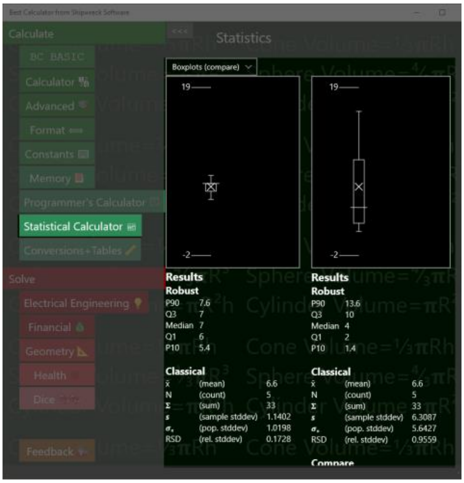
In the regular calculator, 99 × 99 would be 9801. This is (256x38 + 73). Only the bottom byte (the 73) is kept from the calculation. The rest is discarded.

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.



```
Results                                  Results
Robust                                   Robust
P90     7.6                              P90     13.6
Q3      7                                Q3      10
Median  7                                Median  4
Q1      6                                Q1      2
P10     5.4                              P10     1.4

Classical                                Classical
x̄     (mean)             6.6             x̄     (mean)             6.6
N      (count)            5               N      (count)            5
Σ      (sum)              33              Σ      (sum)              33
s      (sample stddev)    1.1402          s      (sample stddev)    6.3087
σₑ     (pop. stddev)      1.0198          σₑ     (pop. stddev)      5.6427
RSD    (rel. stddev)      0.1728          RSD    (rel. stddev)      0.9559

                                         Compare
```

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.

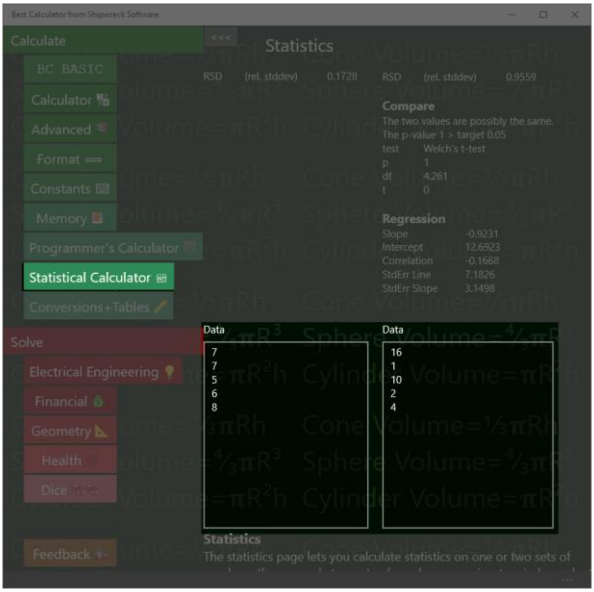
1. **Graphs at the top** present a pictorial view of your data. You can choose the kind of graphs: two independent boxplots, two boxplots for comparison (their Y axes will be set to be the same), an XY Scatter plot, or no graph
2. **Computed results** for the left and right data sets. Both data sets 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.



Statistics

RSD (rel. stddev) 0.1728 RSD (rel. stddev) 0.9559

Compare
The two values are possibly the same.
The p-value 1 > target 0.05

test	Welch's t-test
p	1
df	4.261
t	0

Regression

Slope	-0.9231
Intercept	12.6923
Correlation	-0.1668
StdErr Line	7.1826
StdErr Slope	3.1498

Data	Data
7	16
7	1
5	10
6	2
8	4

Statistics
The statistics page lets you calculate statistics on one or two sets of

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.

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.

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

x̄	(mean)	6.6
N	(count)	5
Σ	(sum)	33
s	(sample stddev)	1.1402
$\boldsymbol{\sigma}_n$	(pop. stddev)	1.0198
RSD	(rel. stddev)	0.1728

Mean

The mean (**x̄**) 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 (**Σ**) 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 $\boldsymbol{\sigma}_n$) 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.

Robust

P90	7.6
Q3	7
Median	7
Q1	6
P10	5.4

Computed robust statistics

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

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	90th	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	75th	The Q3 (third quartile) is the ¾ mark. The box in the boxplot is bounded by the Q# and Q1 points.
Median	50th	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	25th	The Q1 (first quartile) is the ¼ mark
P10	10th	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.

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

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.

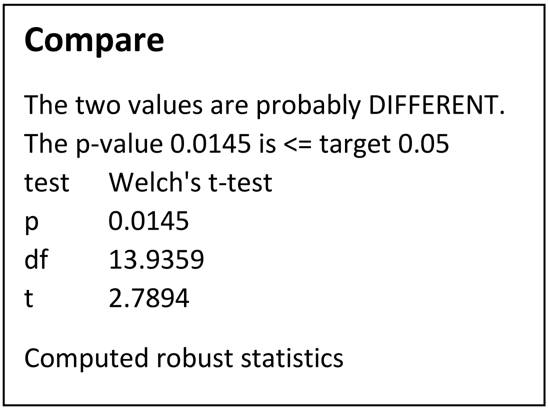
The **correlation** coefficient says how close of a match the data is. If the data isn't correlated at all, the correlation is 0. A negative correlation means that as one value increases, the other decreases (also called a negative correlation). A value of 1 or -1 means that the data is perfectly correlated.

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.

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 through 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)

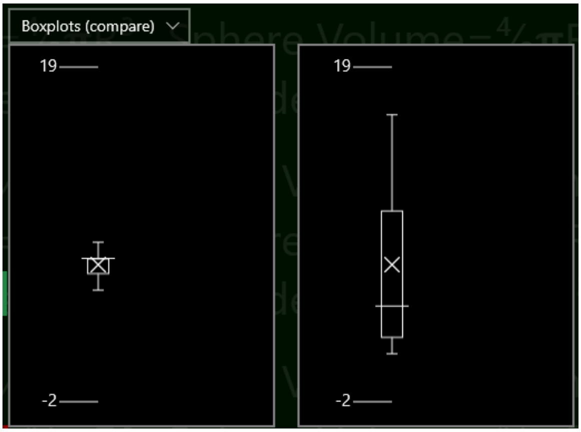
The computed p-value helps answer the question, “are these two data sets likely to be from the “same” kind of data. If the p value computed is small (set to 0.05 in Best Calculator), the two data sets must not be the same and are therefore different. If the p value is large, then no conclusion can be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



Boxplots let you 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 ½ of the data falls within the central box; ¼ is less than and ¼ 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× 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.

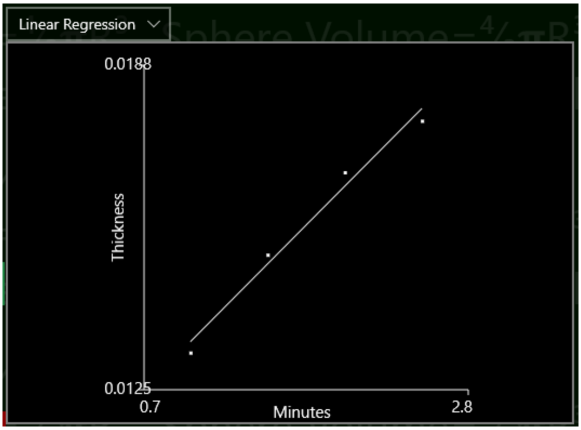
When you're displaying two data sets, you can **compare** the two boxplot or display them as **independent**. In compare mode, the two boxplots are display using the same scale. In independent mode, the two boxplots are display with their own scale.

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.



XY Scatter plot with regression line lets you visualize the linear regression computation

Sample data:

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

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.

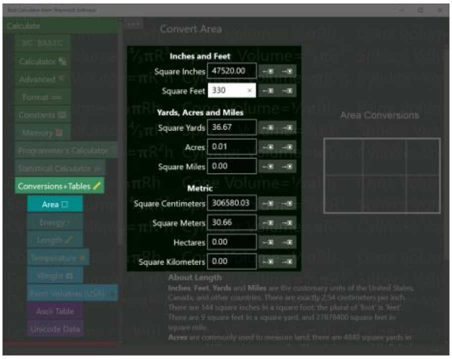
The min and max axis values are picked based on the data and are not settable.

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.



Convert Area

Inches and Feet
Square Inches 47520.00
Square Feet 330

Yards, Acres and Miles
Square Yards 36.67
Acres 0.01
Square Miles 0.00

Metric
Square Centimeters 306580.03
Square Meters 30.66
Hectares 0.00
Square Kilometers 0.00

About Length
Inches, **Feet**, **Yards** and **Miles** are the customary units of the United States, Canada, and other countries. There are exactly 2.54 centimeters per inch. There are 144 square inches in a square foot; the plural of 'foot' is 'feet'. There are 9 square feet in a square yard, and 27878400 square feet in square mile.
Acres are commonly used to measure land; there are 4840 square yards in

Conversions include area, energy, length and more.

22.1 CONVERTING

All of the converters work the same way.

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 ←▣ 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 × 3 =
Tap ‘Conversions and Tables’ and then tap ‘Area’.  Tap the ←▣ 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 →▣ key next to the square feet box. Go to the main calculator; note that it has been set to 360. Key in × 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)

22.8 Farm Volumes (US)

Best Calculator can convert between

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

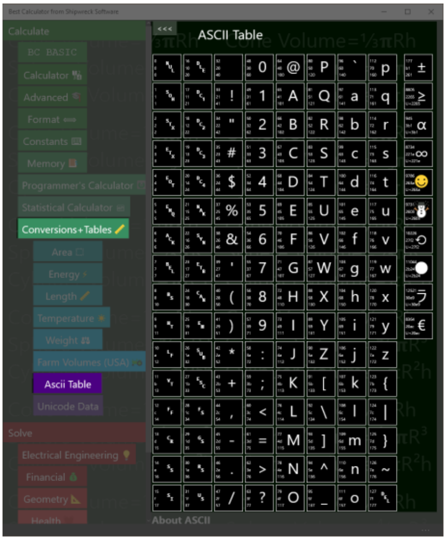
Given a weight in pounds per bushel, Best Calculator will also calculate the weight of an amount in bushels.

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.

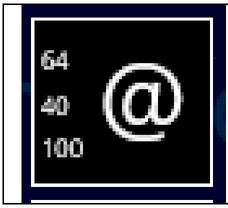
Programmers often need to convert from characters (like "@") into their ASCII equal (the number 64).

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.



	0	@	P	`	p	±
!	1	A	Q	a	q	≥
"	2	B	R	b	r	α
#	3	C	S	c	s	∞
$	4	D	T	d	t	😊
%	5	E	U	e	u	☃
&	6	F	V	f	v	⟲
'	7	G	W	g	w	●
(	8	H	X	h	x	ラ
)	9	I	Y	i	y	€
*	:	J	Z	j	z	
+	;	K	[	k	{	
,	<	L	\	l	|	
-	=	M	]	m	}	
.	>	N	^	n	~	
/	?	O	_	o	[illegible]	

ASCII table



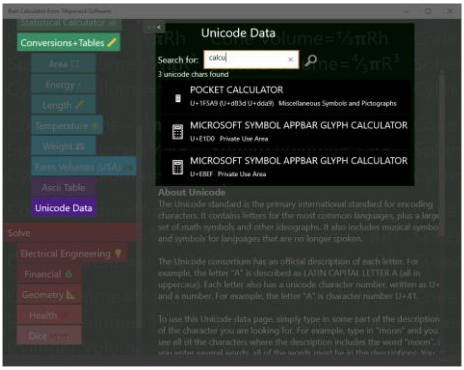
64 is the decimal value of @
40 is the hex value of @
100 is the octal value of @

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 9.0, introduced in 2016



calcu

3 unicode chars found

POCKET CALCULATOR
U+1F5A9 (U+d83d U+dda9) Miscellaneous Symbols and Pictographs

MICROSOFT SYMBOL APPBAR GLYPH CALCULATOR
U+E1D0 Private Use Area

MICROSOFT SYMBOL APPBAR GLYPH CALCULATOR
U+EBEF Private Use Area

About Unicode

The Unicode standard is the primary international standard for encoding characters. It contains letters for the most common languages, plus a large set of math symbols and other ideographs. It also includes musical symbo and symbols for languages that are no longer spoken.

The Unicode consortium has an official description of each letter. For example, the letter "A" is described as LATIN CAPITAL LETTER A (all in uppercase). Each letter also has a unicode character number, written as U+ and a number. For example, the letter "A" is character number U+41.

To use this Unicode data page, simply type in some part of the description of the character you are looking for. For example, type in "moon" and you see all of the characters where the description includes the word "moon".

Find and copy Unicode characters including emoji

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

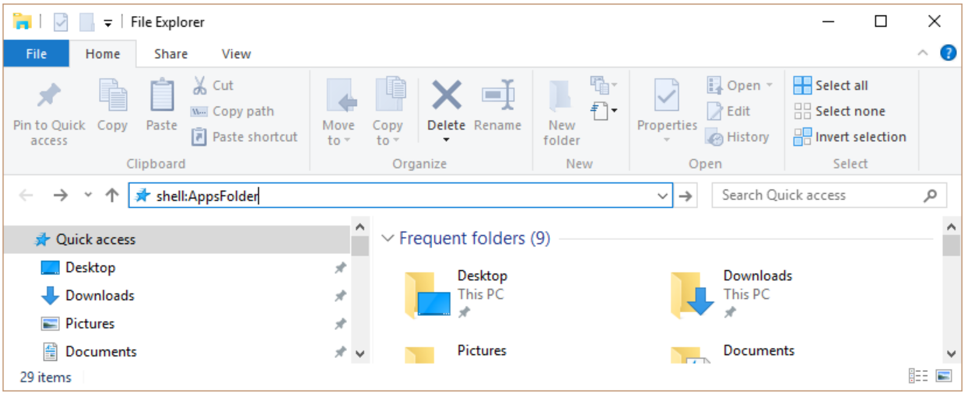
Right-click on a result to bring up the app bar. Tap the clipboard to copy the character to the clipboard.

25 Advanced Windows Features

25.1 Shortcut on the Desktop

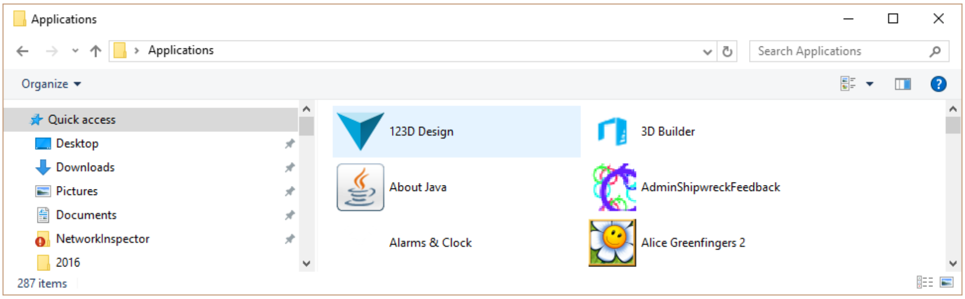
You can add Best Calculator as a shortcut on your desktop.

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



shell:AppsFolder

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.

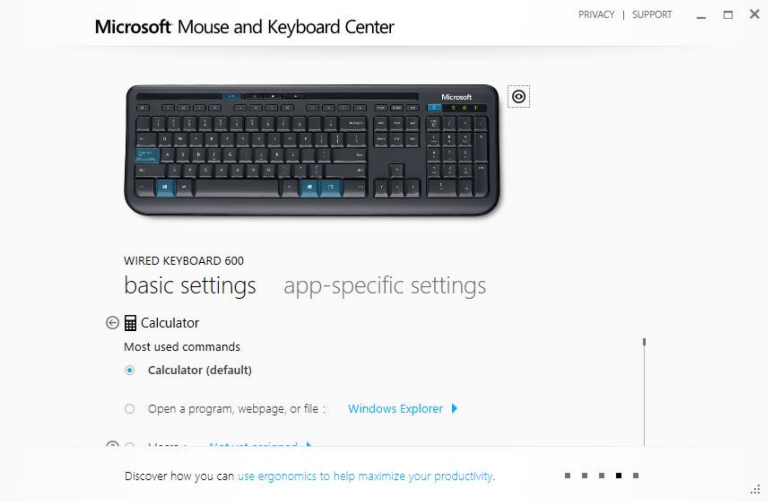
25.2 SET AS THE CALCULATOR KEY

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.



WIRED KEYBOARD 600

basic settings app-specific settings

Calculator

Most used commands

● Calculator (default)

○ Open a program, webpage, or file : Windows Explorer

Discover how you can use ergonomics to help maximize your productivity.

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

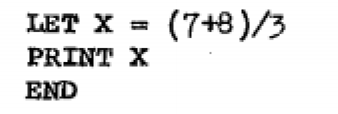
```
explorer shell:AppsFolder\48425ShipwreckSoftware.BestCalculator_jh2negtepkzpr!App
```

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 A BRIEF HISTORICAL NOTE

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

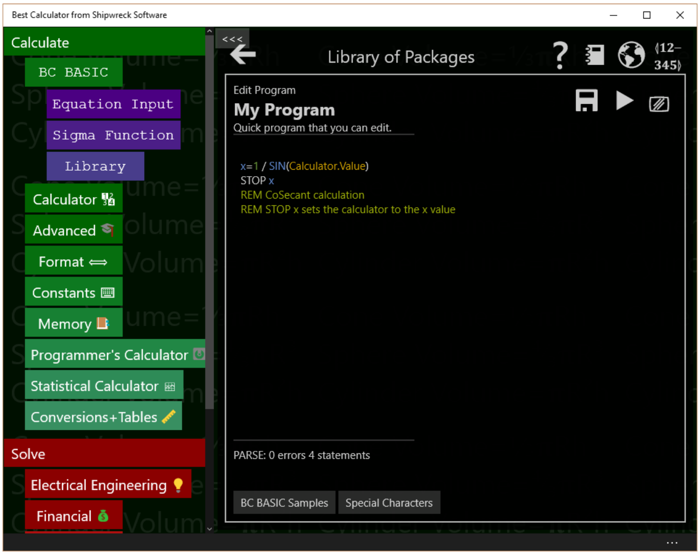
Calculator BASIC lets you program your own simple functions into the Best Calculator, extending its abilities to exactly match your needs. Everyone who's a specialist has the same problem with typical calculators: there's some standard calculation in your profession, but no calculator includes those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

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.

27 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.



Edit Program

My Program

Quick program that you can edit.

```
x=1 / SIN(Calculator.Value)
STOP x
REM CoSecant calculation
REM STOP x sets the calculator to the x value
```

PARSE: 0 errors 4 statements

27.1 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

27.2 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.

27.3 How can I print some text?

Use the PRINT statement like this:

```
PRINT “Calculation complete!”
```

27.4 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)

27.5 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
```

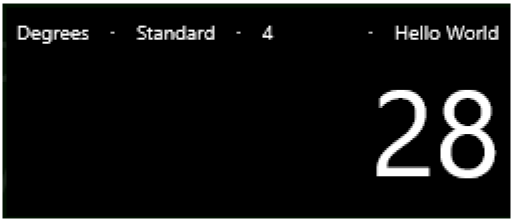
27.6 How can I write to the calculator?

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”
```

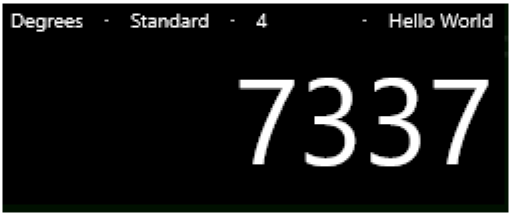
Which then show up on the calculator results windows



```
Degrees - Standard - 4 - Hello World
28
```

Use Calculator.Value to get or set the result value

```
Calculator.Value = 7337
```



Degrees · Standard · 4 · Hello World

7337

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.

27.7 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

27.8 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.

27.9 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 ◁12 345▷ (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 ▨ (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.

27.10 How can I learn more?

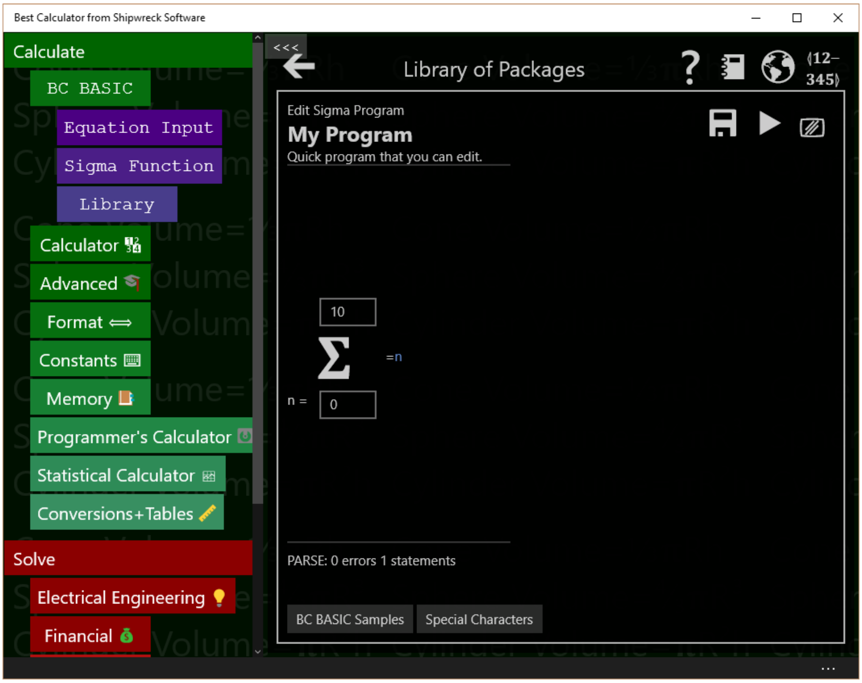
The Reference: Language Basics section tells you everything that makes up a valid BC BASIC program. As a helpful reminder, press the BC BASIC Samples key

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.

28 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.



Edit Sigma Program

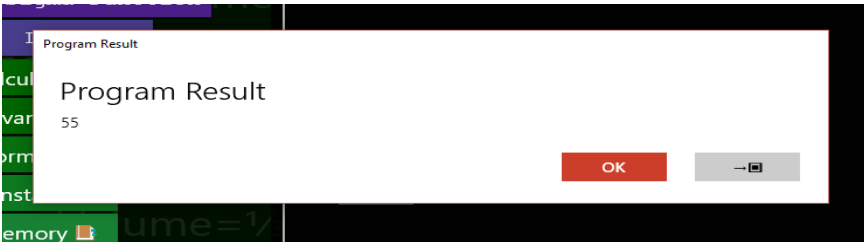
My Program

Quick program that you can edit.

$$\sum_{n=0}^{10} n$$

PARSE: 0 errors 1 statements

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



Program Result

Program Result

55

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**$^{\mathbf{series}}$ value and divides by **series!**. Odd parts of the sequence are supposed to be subtracted from the sequence, so those are negated.

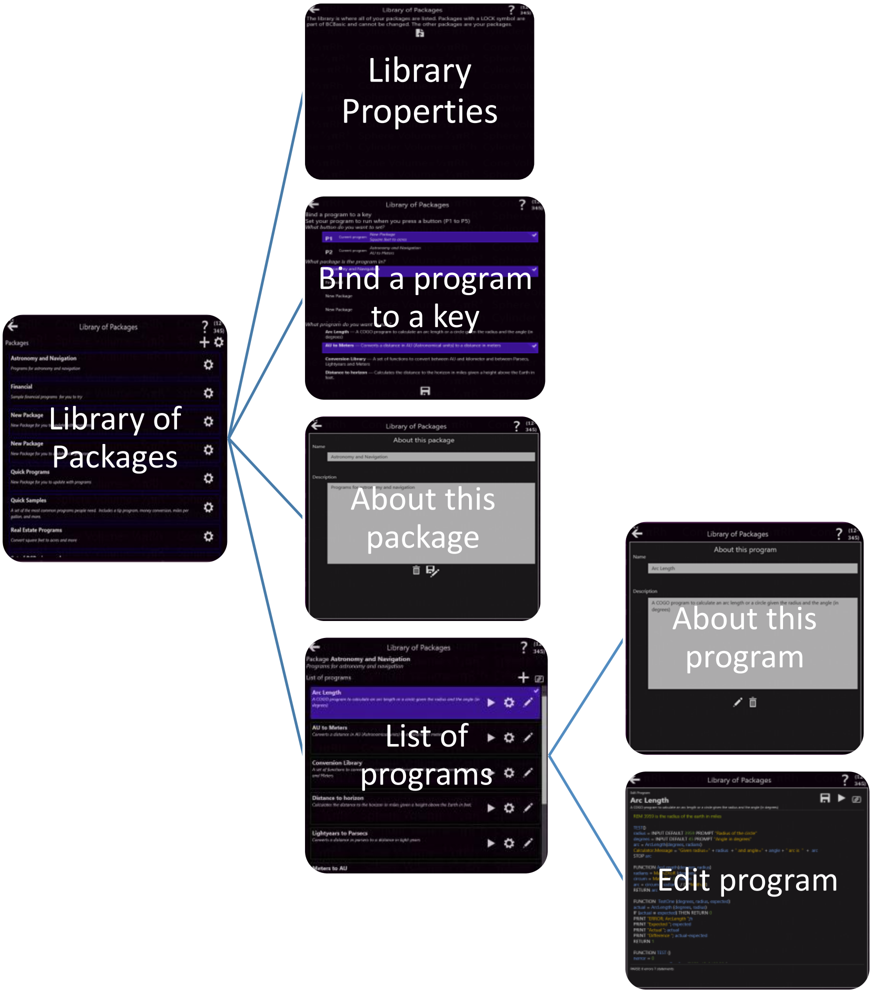
The x value is taken from the **Calculator.Value** number from the calculator display.

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

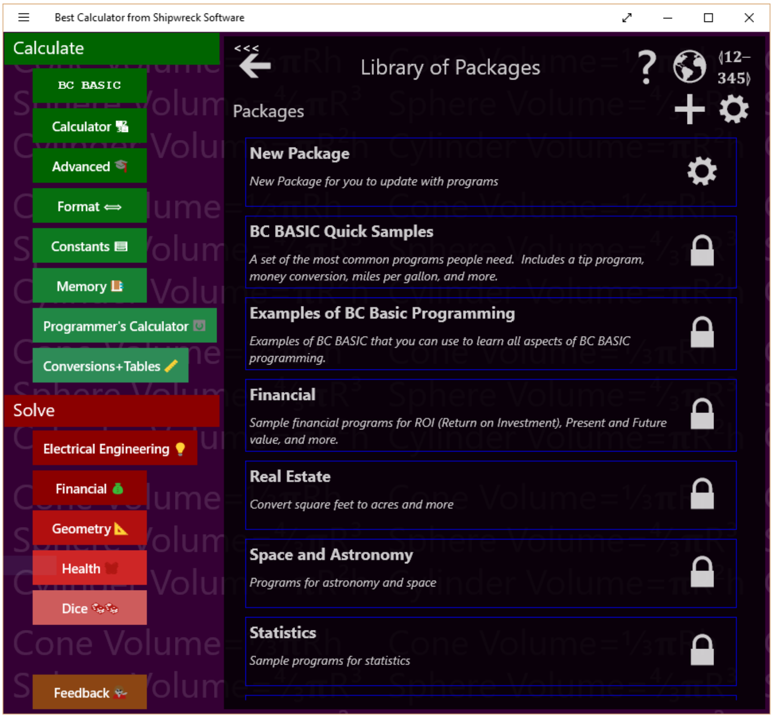
29 WHAT ALL CAN YOU DO IN THE BC BASIC ENVIRONMENT?

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

29.1 ALL THE MAIN EDIT DIALOGS



29.2 Library of Packages



Packages

New Package
New Package for you to update with programs

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

Examples of BC Basic Programming
Examples of BC BASIC that you can use to learn all aspects of BC BASIC programming.

Financial
Sample financial programs for ROI (Return on Investment), Present and Future value, and more.

Real Estate
Convert square feet to acres and more

Space and Astronomy
Programs for astronomy and space

Statistics
Sample programs for statistics

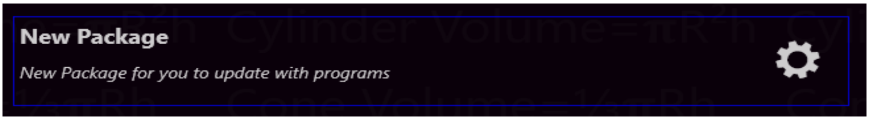
Library of Package is the first library dialog you see. It lists all of the packages that you can run, examine, or change.

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

Control	How and when to use it
← Back arrow	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. Don't worry! None of your changes are lost. Just press the BC BASIC key to see the Library of Packages dialog again The little "chevron" arrow is a Best Calculator arrow; it hides the Best Calculator menu of calculators.

? Help	Displays the main HELP PDF page using the default PDF reader (often a web browser).
📓 Buy the manual	Goes to Amazon.com. You can buy a copy of the Best Calculator manual from Amazon.
🌍 Web site	Goes to the Best Calculator web site
◁12 345▷ Bind Key	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.
+ Create new package	Creates a new package. The new package gets a name which you should change. Use the GEAR icon (⚙) to change the name.
⚙ Properties	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.



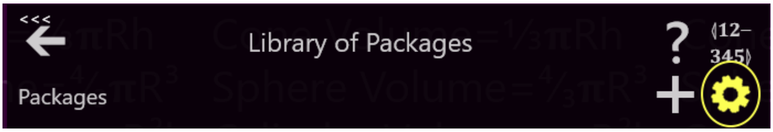
New Package

New Package for you to update with programs

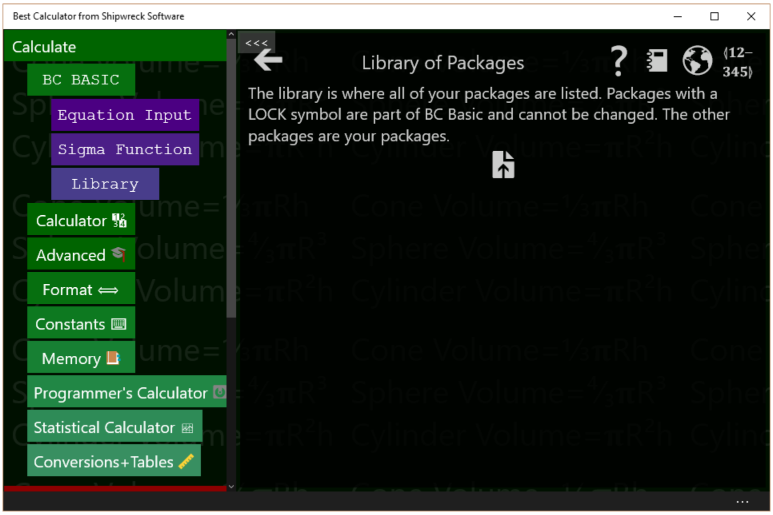
Control	When and how to use it
Tap on a package	Brings up the List of programs for that package. From the list you can add new programs and edit and run your programs.
⚙ Properties	Bring up the About this package dialog. From that dialog you can change the name and description for a package.

29.3 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.

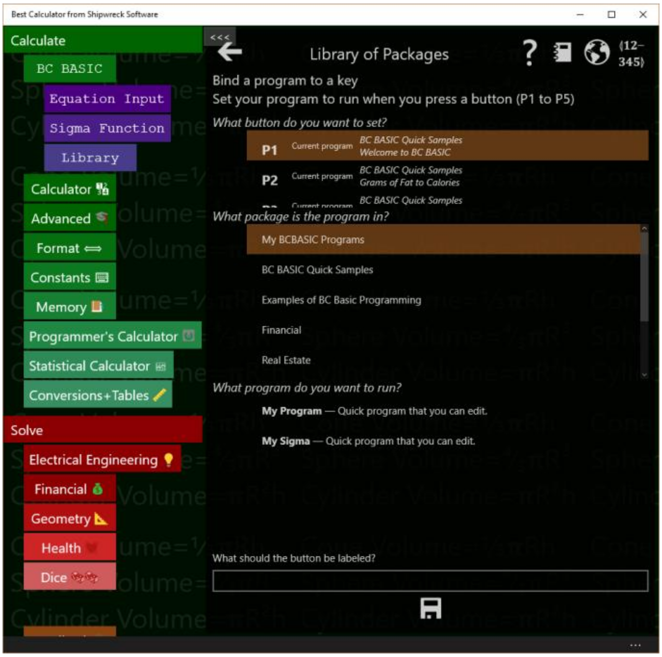


The library is where all of your packages are listed. Packages with a LOCK symbol are part of BC Basic and cannot be changed. The other packages are your packages.

In the Library properties there is a single control

Control	When and how to use it
Import	Brings up the Import dialog. Use this to import (read in) new BC BASIC packages from your job, your school, your friends, or even the internet.

29.4 BIND A PROGRAM TO A KEY



Bind a program to a key

Set your program to run when you press a button (P1 to P5)

What button do you want to set?

P1	Current program	*BC BASIC Quick Samples* *Welcome to BC BASIC*
P2	Current program	*BC BASIC Quick Samples* *Grams of Fat to Calories*
	Current program	*BC BASIC Quick Samples*

What package is the program in?

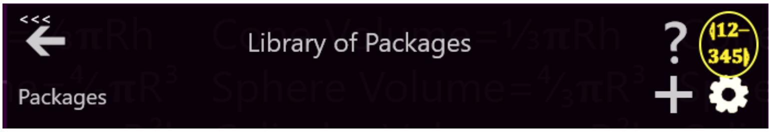
- My BCBASIC Programs
- BC BASIC Quick Samples
- Examples of BC Basic Programming
- Financial
- Real Estate

What program do you want to run?

- **My Program** — Quick program that you can edit.
- **My Sigma** — Quick program that you can edit.

What should the button be labeled?

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



There are several programmable key 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.

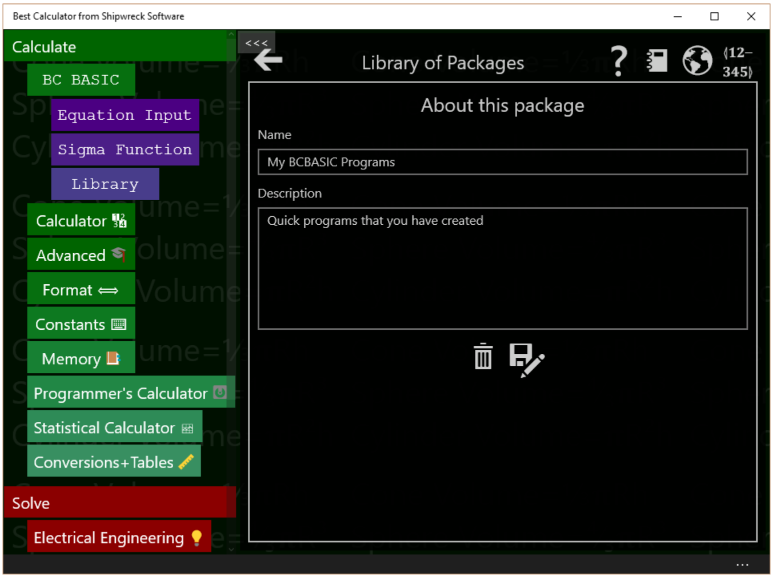
The third question is *What program do you want to run?* Tap the program to run.

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.

29.5 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.



About this package

Name

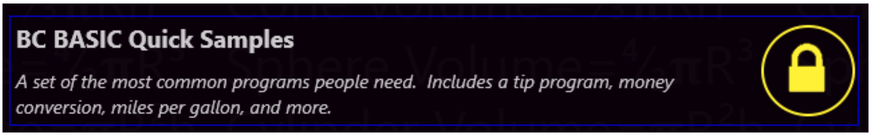
My BCBASIC Programs

Description

Quick programs that you have created

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).

polYou 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.



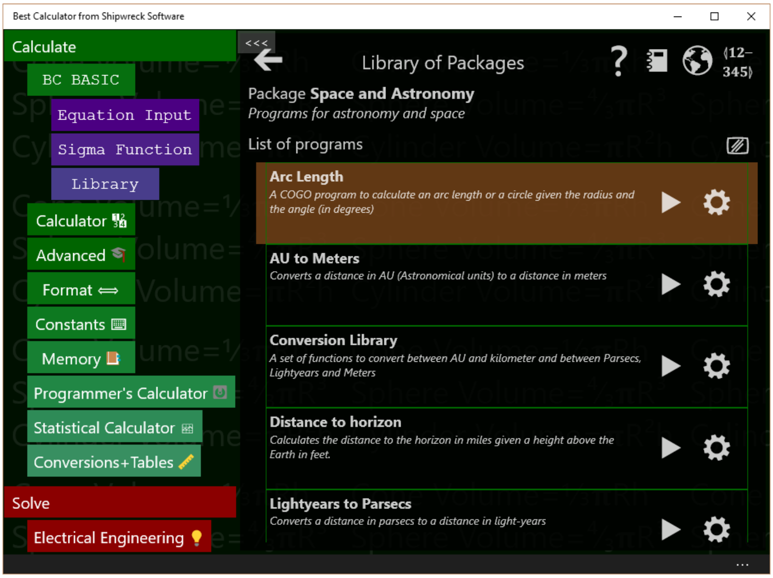
BC BASIC Quick Samples

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.
🗑 Delete	Deletes the package. Once deleted, you will not be able to bring the package back. You will be prompted to confirm the delete.
💾 Save As	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.

29.6 List of Programs

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



Package **Space and Astronomy**
Programs for astronomy and space

List of programs

Arc Length
A COGO program to calculate an arc length or a circle given the radius and the angle (in degrees)

AU to Meters
Converts a distance in AU (Astronomical units) to a distance in meters

Conversion Library
A set of functions to convert between AU and kilometer and between Parsecs, Lightyears and Meters

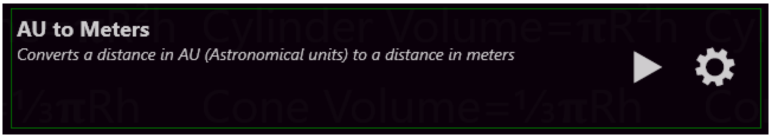
Distance to horizon
Calculates the distance to the horizon in miles given a height above the Earth in feet.

Lightyears to Parsecs
Converts a distance in parsecs to a distance in light-years

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.

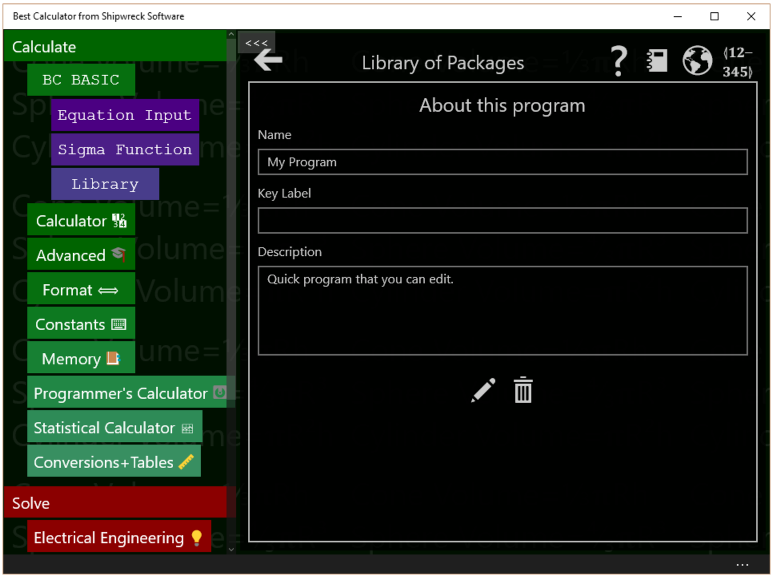


AU to Meters

Converts a distance in AU (Astronomical units) to a distance in meters

Control	**When and how to use it**
Tap on a program	Does nothing in particular ☺
▶ Run program	Runs the program from the start
⚙ Properties	Displays the About this program dialog. From that dialog you can edit the program name and description.
✎ Edit program	Display the Edit Program dialog. From that dialog you can edit and run the program. Packages that come with Best Calculator cannot be modified. For those program, the Edit key will show you the source code for the program but will not let you change it.

29.7 ABOUT THIS PROGRAM



About this program

Name

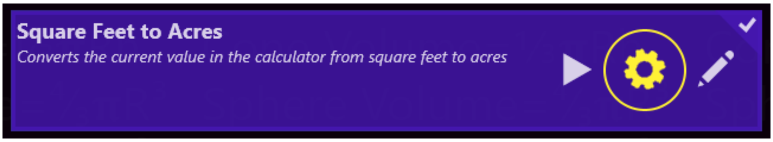
My Program

Key Label

Description

Quick program that you can edit.

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.

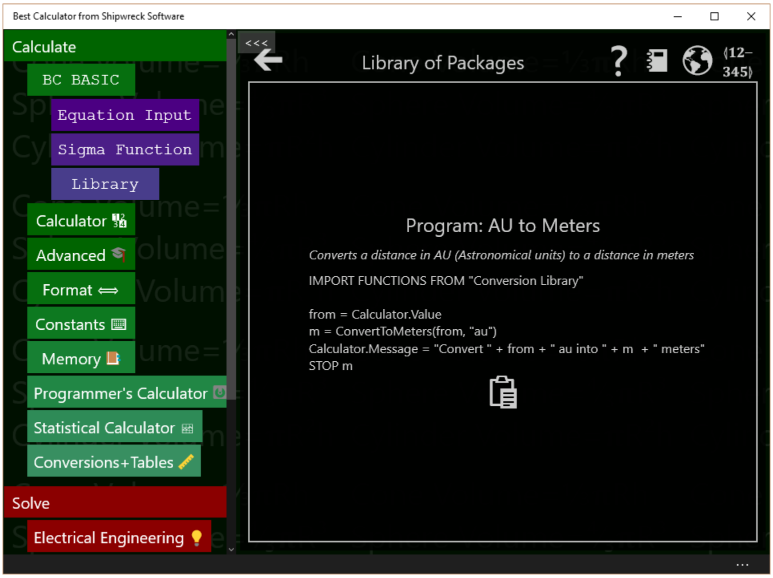

Square Feet to Acres

Converts the current value in the calculator from square feet to acres

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.



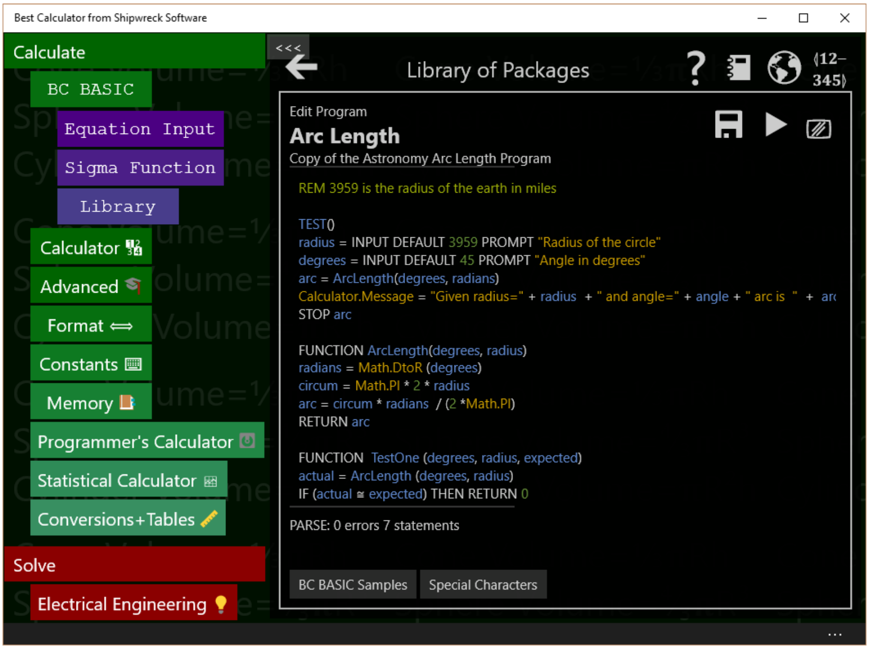
Program: AU to Meters

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
```

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

29.8 Edit Program



Arc Length

Copy of the Astronomy Arc Length Program

```
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  "  +  ar
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
```

PARSE: 0 errors 7 statements

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.

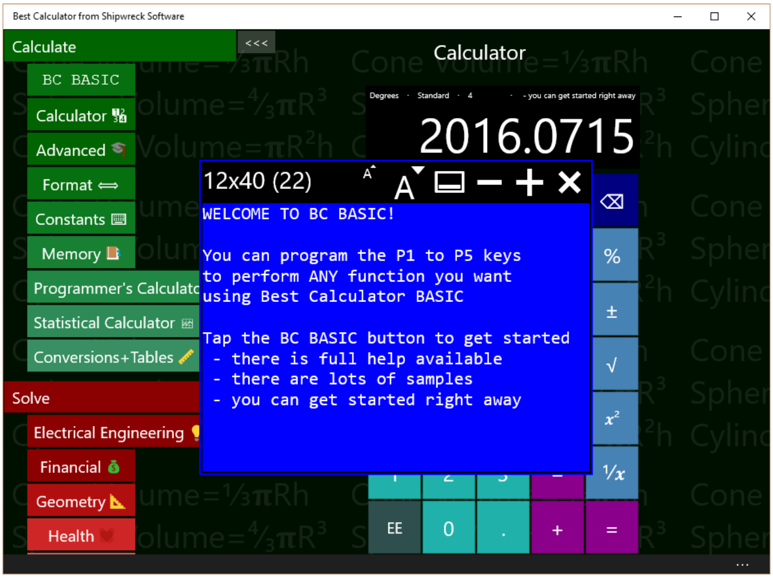
Control	**How and when to use it**
💾 Save	Saves your changes. Your changes are automatically saved when you press RUN. Your program is saved as part of the package file; this is managed for you.
▶ Run program	Runs the program from the start. Some common errors when you press RUN but your program does not appear to run are: 1. Your program might have a syntax or other error that prevents it from running. You will can tell because the parse indicator will show an error

	2. You 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.
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 you program compiles. BC BASIC automatically compiles your program as you type it, and tells you of any syntax errors. The editor uses *syntax coloring* 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.

There are two 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 key. Press F5 to start the program running.

29.9 OUTPUT SCREEN



```
WELCOME TO BC BASIC!

You can program the P1 to P5 keys
to perform ANY function you want
using Best Calculator BASIC

Tap the BC BASIC button to get started
 - there is full help available
 - there are lots of samples
 - you can get started right away
```

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
12x40 (26) Screen Size	The output screen always tells you how large the screen is and the font size. Screen sizes include 12x20, 12x40, 16x60 and 24x80
A▲ Smaller font	Reduces the font size, automatically making the output screen smaller.
▼ A	Increases the font size, automatically making the output screen larger

Larger font	
▭ Display/hide console	Toggles the console portion of the output screen on and off. Unlike the output screen, the console is a long scrolling list of output.
− Fewer characters on screen	Reduces the number of characters on the screen.
+ More characters on screen	Increases the number of characters on the screen.
✕ Close	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 BASIC LANGUAGE REFERENCE

30.1 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().

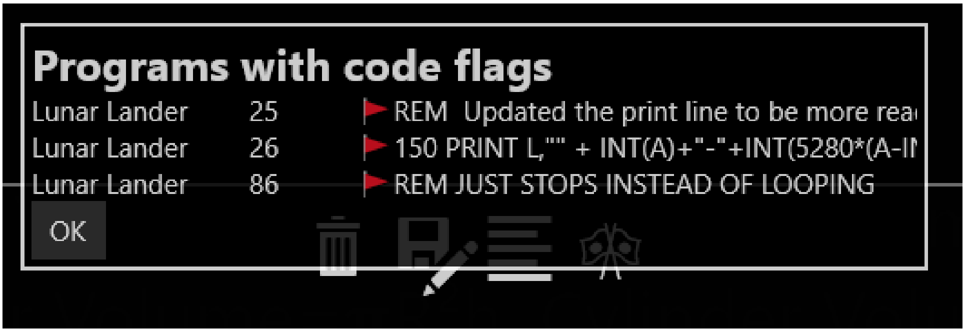
Lines are technically a sequence of characters that ends with a \n, \r or \v. The \v (vertical tab) is sometimes used by Microsoft Word when cut-and-pasting text.

30.2 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.



Programs with code flags

Lunar Lander	25	🚩 REM Updated the print line to be more rea
Lunar Lander	26	🚩 150 PRINT L,"" + INT(A)+"-"+INT(5280*(A-I
Lunar Lander	86	🚩 REM JUST STOPS INSTEAD OF LOOPING

30.3 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 (lie, “apple”).

30.3.1 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)

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.

30.3.2 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.

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.

30.3.3 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”.

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 are 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.IsErrror)
    PRINT value.ErrorCode
    PRINT value.ErrorString
END IF
```

30.4 <EXPRESSION> OVERVIEW

30.4.1 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.

30.4.2 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. Unlick 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 ¼ 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.

30.4.3 negate, power, root prefix and postfix operators

Operator	Explanation and Sample
^{2} ^{3} ^{4}	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 ** operator Example: 5^2 is 25 because 5*5 is 25 Math.Pow(5, 2) is also 25 5**2 is also 25
√ ∛ ∜	Square root, cube root and fourth root operators. You can also use the Math.Pow () function or the ** operator Example: √64 is 8 because 8*8 is 64 Math.Pow(64, 1/2) is also 8 64**0.5 is also 8
-	Unary minus converts any number to its negative Example: 12 + -3 is 9 because 12 – 3 is 9

30.4.4 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	** is the “raise to the power” operator Example: 2 ** 6 is 64 because 2*2*2*2*2*2 is 64

		Historical note: early versions of BC Basic includes a "root finding" operator. This proved to be confusing in practice, and has been removed.
* /	9	* is the computer sign for multiply / is the standard computer sign for divide Examples: 3 * 4 is 12 10 / 2 is 5
+ -	6	+ is the standard computer sign for addition or string concatenation - is the standard computer sign for subtraction 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 Extra bonus: The Unicode character set designates three characters that are commonly used for minus signs: HYPHEN-MINUS (U+2D), the normal minus sign MINUS SIGN (U+2212) EN DASH (U+2013) 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.
< <= = >= > <> ≅	5	The normal set of operators for less than, less-than-or-equal, and so on. The <> operator is for "not equals".

		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 equal using the CurrentCultureIgnoreCase flag. Note that two numbers which are different signs are never considered approximately equal even if they are both really, really close to zero.
NOT	4	Inverts the logical value of its argument. Example IF NOT A=B THEN <statement> will do the statement if A is NOT equal to B. **Note on logical values:** 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.
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		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. Example:

		birth_year = INPUT DEFAULT 1967 PROMPT “Enter your birth year: ”

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
```

30.4.5 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.

30.5 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.

30.5.1 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

30.5.2 Logarithm and Power functions LN EXP SQR

Function	Notes
EXP (value)	Calculates the value e^{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).

30.5.3 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)

30.6 STRING FUNCTIONS (LEFT, MID, RIGHT, LEN, CHR$, CODE, VAL)

30.6.1 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 1. 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

30.6.2 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.

30.6.3 CHR, CODE

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”`.

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"
```

30.6.4 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")
```

30.7 <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.

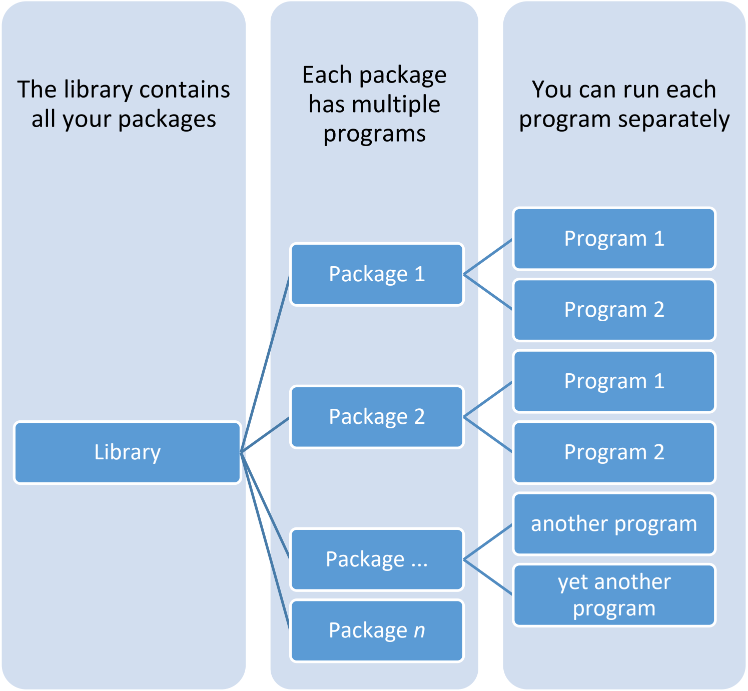
Statements are separated by new-lines; a newline is one or more of carriage-return, new-line, or vertical-tab. (Pressing ALT-newline in Word can separate lines with vertical tabs). You can continue a statement from one line to another by placing a visible enter symbol (↲, U+21B2) at the end of the line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

30.8 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).

30.8.1 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).

30.9 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. 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.

30.9.1 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.

30.9.2 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 JSON 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.

30.10 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
√	Square Root U+221A	C = √ (A^2 + B^2)
∛	Cube Root U+221B	X = ∛Y
∜	Fourth Root U+221C	W = ∜ (X+Y)
²	Superscript 2 U+B2	$C=X^2$
³	Superscript 3 U+B3	$C=X^3$
⁴	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
≅	Approximately equal to U+2245	Helpful to compare floating point numbers.
“ ”	Left and right double quotation marks U+201C and U+201D	Smart Quotes
\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
↲	Downwards arrow with tip leftwards U+21B2	Line continuation: use at the end of a line to continue onto the next line.
↵	Downwards arrow with corner leftwards U+21B5	Line continuation
⤶	Arrow pointing downwards then curving leftwards U+2936	Line continuation

31 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 ()
```

31.1 [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.

31.2 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.

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.

31.3 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.

31.4 DIM <NAME> ([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 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
```

31.5 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

31.5.1 Count property

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

31.5.2 Max and Min properties

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

31.5.3 MaxOf(column) and MinOf(column) properties

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

31.5.4 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.

31.5.5 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.

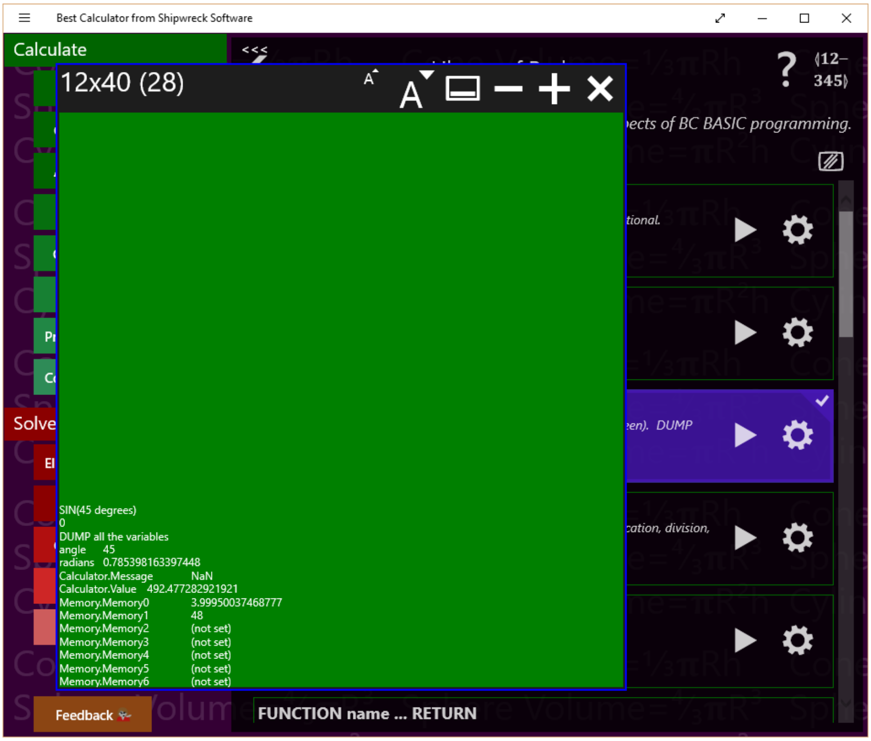
31.6 DUMP

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
RADIANS = Math.DtoR (ANGLE)

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

CONSOLE "DUMP all the variables"
DUMP
```

The variables are then printed to the scrolling console.



```
SIN(45 degrees)
0
DUMP all the variables
angle     45
radians   0.785398163397448
Calculator.Message          NaN
Calculator.Value   492.477282921921
Memory.Memory0           3.99950037468777
Memory.Memory1           48
Memory.Memory2           (not set)
Memory.Memory3           (not set)
Memory.Memory4           (not set)
Memory.Memory5           (not set)
Memory.Memory6           (not set)
```

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

31.7 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.

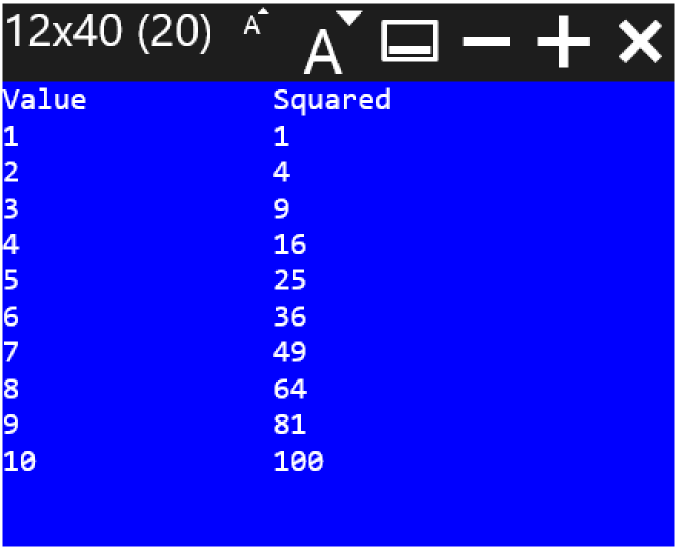
Common errors:

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:

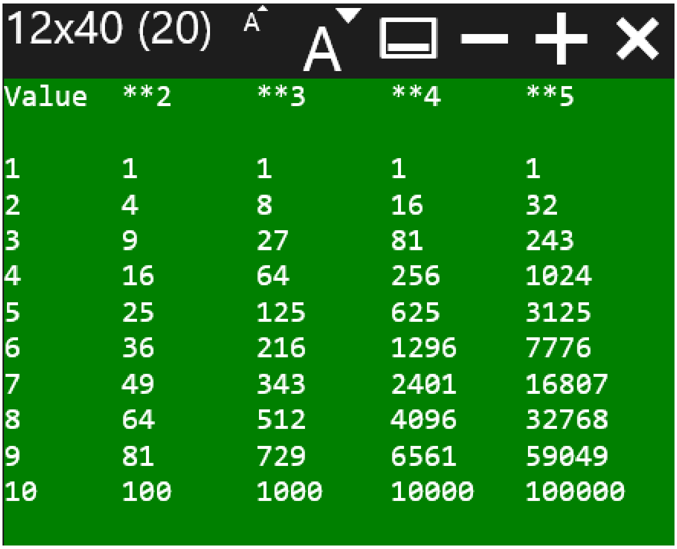
```
PRINT “Value”, “Squared”
FOR I=10 TO 1 STEP –1
PRINT I, I**2
NEXT I
```

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
```

31.8 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.

31.9 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
END
```

31.10 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
```

31.11 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.

31.12 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"
```

31.13 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"
END IF
```

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

31.14 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.

31.15 INPUT <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.

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”
```

31.16 (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.

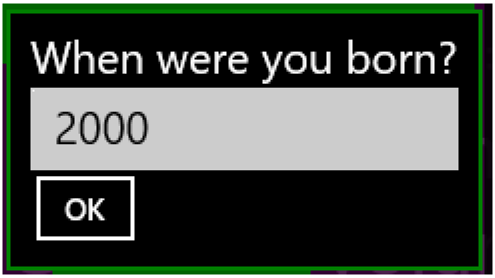
Variables start with a letter, and then can be any combination of letters, digits, and underscores. Variables are case sensitive; myage is different from MYAGE and is different from myAge. Expression can include requests for user input.

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

31.17 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.

31.18 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 of 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.

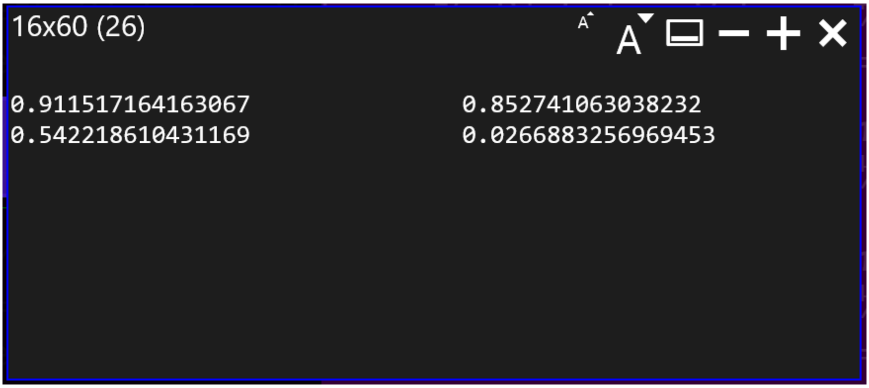
31.19 RAND <SEED> & THE RND VALUE

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:



16x60 (26)

```
0.911517164163067               0.852741063038232
0.542218610431169               0.0266883256969453
```

Technical details

BC BASIC provides access to a single pseudo-random number generator; the stream of values is determined entirely 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.

31.20 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.

31.21 STOP 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 subroutines (called by using GOSUB) at the end. A STOP statement is placed after the main logic, and before the subroutines.

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 prints 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 version of BASIC including the first Dartmouth BASIC program (see the historical note).

32 Extensions Reference

BC BASIC includes *extensions* to the core BASIC language. The extensions let you

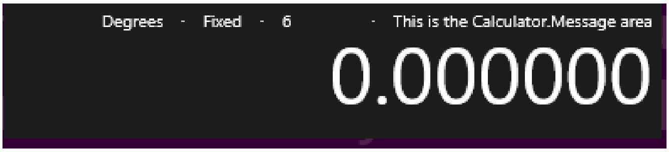
- Set and get values from the calculator screen
- Perform advanced math
- Set and get memory values that persist from one session to another and roam to your other computers
- Get information about the screen

32.1 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”.



Degrees - Fixed - 6 - This is the Calculator.Message area

0.000000

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”.

32.2 DateTime Extension

The DateTime extension is designed to let you add time stamps to your date.

32.2.1 DateTime.GetNow()

The DateTime.GetNow() method returns a filled-in DateTime object set to the current time.

32.2.2 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 as elapsed.

Example:

```
startTime = DateTime.GetNow()
REM now perform some calculations
endTime = DateTime.GetNow()
delta = endTime.Subtract (startTime)
PRINT “That took ”, delta, “ seconds”
```

32.2.3 Hour, Minute, Second, Year, Month, Day and DayOfWeek properties

DateTime includes a number of properties to get the current date and time split into the component parts (hour, minute, etc.).

32.2.4 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 23rd, 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.

32.2.5 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.

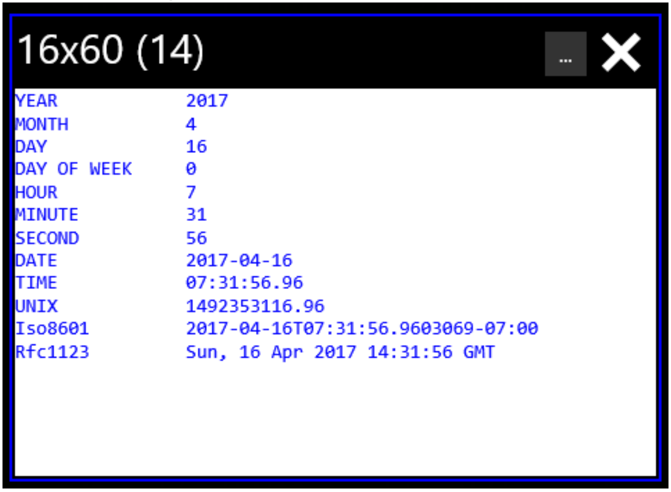
32.2.6 DateTime.AsTotalSeconds

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 1st, 1970.

32.2.7 Example of all the 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
```



```
YEAR            2017
MONTH           4
DAY             16
DAY OF WEEK     0
HOUR            7
MINUTE          31
SECOND          56
DATE            2017-04-16
TIME            07:31:56.96
UNIX            1492353116.96
Iso8601         2017-04-16T07:31:56.9603069-07:00
Rfc1123         Sun, 16 Apr 2017 14:31:56 GMT
```

In the example, the code was run on 2017-04-16 at 7:31 in the morning.

32.3 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.

32.3.1 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))
```

32.3.2 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 (Coma-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
```

32.3.3 file=File.WritePicker(), file.WriteLine, file.WriteText() 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.WriteText (String.Escape(“csv”, data))
```

32.4 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 .

32.4.1 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 "ErrorString", result.ErrorString
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
```

32.4.2 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
```

32.5 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.

32.5.1 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

32.5.2 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 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.) 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.

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)

32.5.3 Logarithm and power functions (Math.Log, Math.Exp, and more)

Function	Notes
Math.Exp(value)	Calculates the value e^{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 $\log_{base}$ 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 * Math.Log2(12) = 717 bits; divide by 8 to 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.

32.5.4 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.

32.5.5 Math.PI, Math.E and Math.NaN values

Two constants, Math.PI and Math.E are available from the Math extension.

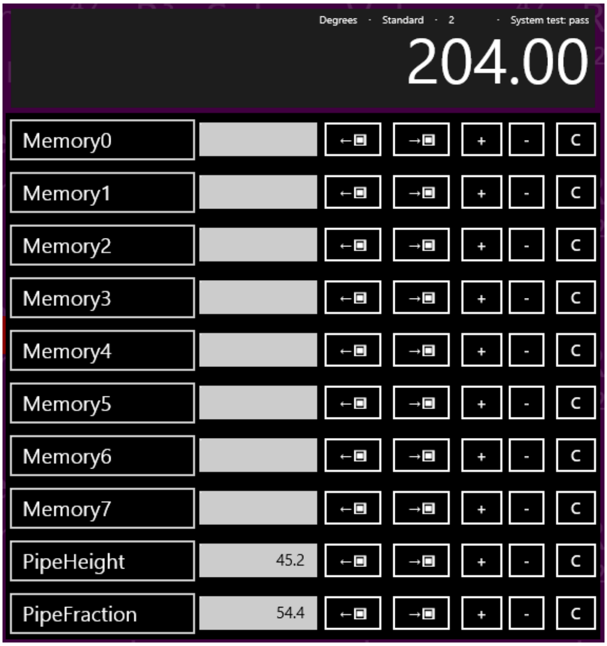
Example of using Math.PI and Math.E:

```
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.

32.6 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).



Degrees · Standard · 2 · System test: pass

204.00

Memory0	
Memory1	
Memory2	
Memory3	
Memory4	
Memory5	
Memory6	
Memory7	
PipeHeight	45.2
PipeFraction	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.

32.6.1 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.

32.6.2 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
```

32.6.3 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 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
```

32.7 Screen Extension

The Screen extension gives you additional information about the graphics screen.

32.7.1 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.

32.7.2 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.

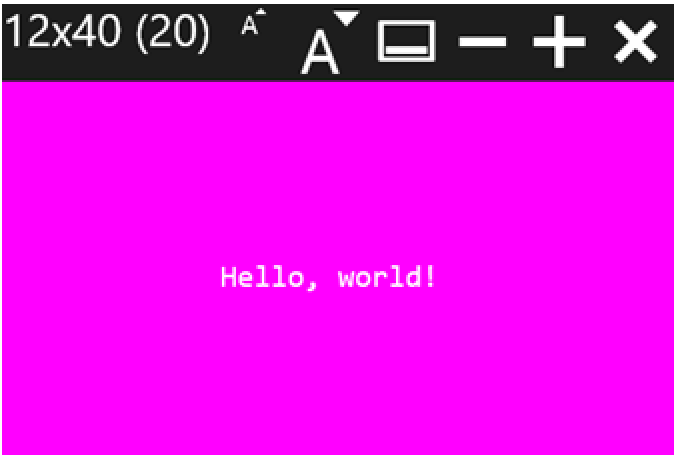
32.7.3 Screen.H and Screen.W Extension

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.



```
Hello, world!
```

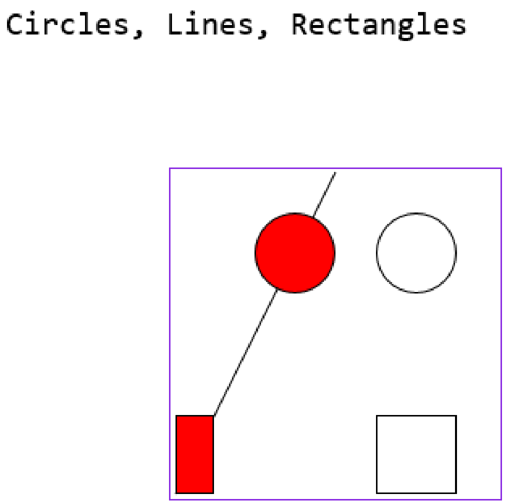
32.8 Screen.Graphics() Extension

The Screen extension includes a Graphics() method that creates 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.

32.8.1 Circle(X, Y, radius) Line(X1, Y1, X2, Y2) Rect(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
```

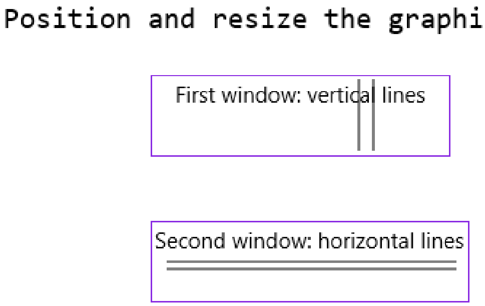
The program first creates a graphics windows (`g = Screen.Graphics()`) and then draws a line, a filled rectangle and circle and a non-filled rectangle and circle.

32.8.2 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.



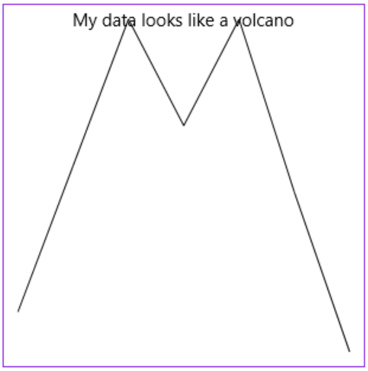
```
Position and resize the graphics windows
```

32.8.3 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)
```

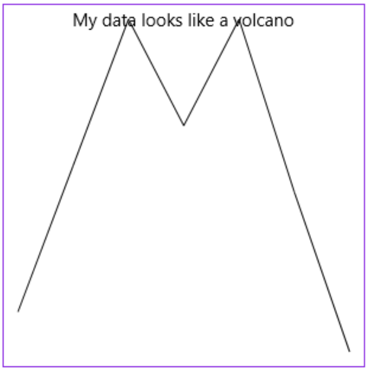


The GraphY() automatic graph takes an array (from the DIM statement) that contains Y (up-and-down) values. The X value for the graph is taken directly from the index of the data (1, 2, 3, and so on).

32.8.4 GraphY(data)

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).

32.8.5 Updating Data with graph.Update() and PAUSE

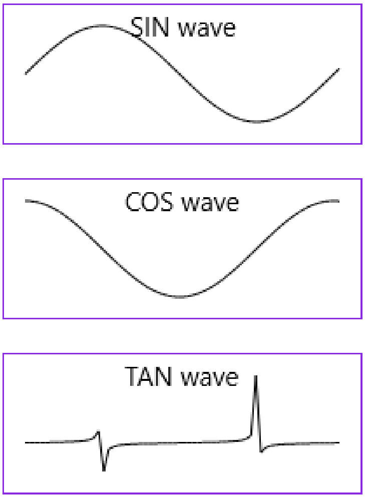
When you update the data in the DIM'd array that you passed into GraphY, the graph will automatically update. The update happens when you do a PAUSE command or when you call graph.Update()

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 101st 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.

32.9 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).

32.9.1 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.

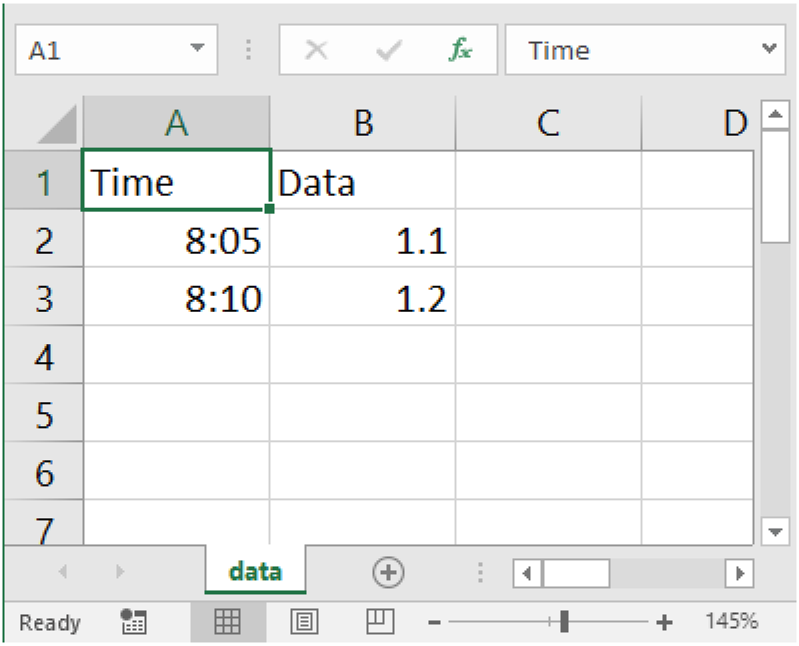
The escape is most commonly used with an array. An array (made with a statement like `DIM data()`)will be converted into a single line of a CSV file.

```
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.WriteText (line)

data[1] = "8:05"
data[2] = 1.1
line = String.Escape ("csv", data)
file.WriteText (line)

data[1] = "8:10"
data[2] = 1.2
line = String.Escape ("csv", data)
file.WriteText (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:



	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.

32.9.2 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 the conventions of RFC 7159 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
}
```

32.9.3 String.Parse(“csv”, <data string>)

Use String.Parse(“csv”, <data string>) to convert CSV-encoded file data into a data array.

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.

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.ReadAll()
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
```

32.9.4 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
```

33 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. Unlike the normal Best Calculator, the Best Calculator, IOT edition is a paid app. However, it does have a free trial so that you can make sure that it’s suitable for your device. You should have a basic understanding of Bluetooth devices to use these features.

33.1 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 “”
END
```

The `devices = Bluetooth.Devices ()` line gets all the paired Bluetooth 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 paid version of Best Calculator, IOT edition 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

33.2 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 “”
END
```

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.

33.2.1 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 “”
END
```

33.3 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.

33.3.1 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.

33.3.2 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.

33.3.3 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.

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.

33.3.4 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.

33.4 SELECTING A DEVICE WITH PICKDEVICESNAMES AND MORE

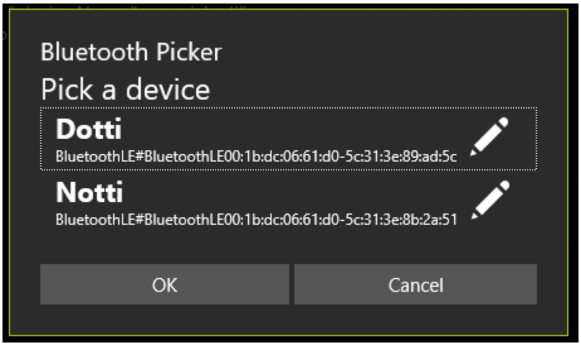
Often you need to be able to pick a specific device to control. For example, you might have several of the NOTTI illuminated devices. You want to pick which NOTTI device you want to control.

33.4.1 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("*ot*")
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.



Bluetooth Picker

Pick a device

Dotti
BluetoothLE#BluetoothLE00:1b:dc:06:61:d0-5c:31:3e:89:ad:5c

Notti
BluetoothLE#BluetoothLE00:1b:dc:06:61:d0-5c:31:3e:8b:2a:51

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. 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.

33.4.2 Bluetooth.DevicesName(<name pattern>)

Bluetooth.DevicesName(<name pattern>) lets you access the same list of devices that the PickDevicesName method shows the use. You can use this when you need to perform the same action on multiple devices.

33.4.3 Bluetooth.Devices()

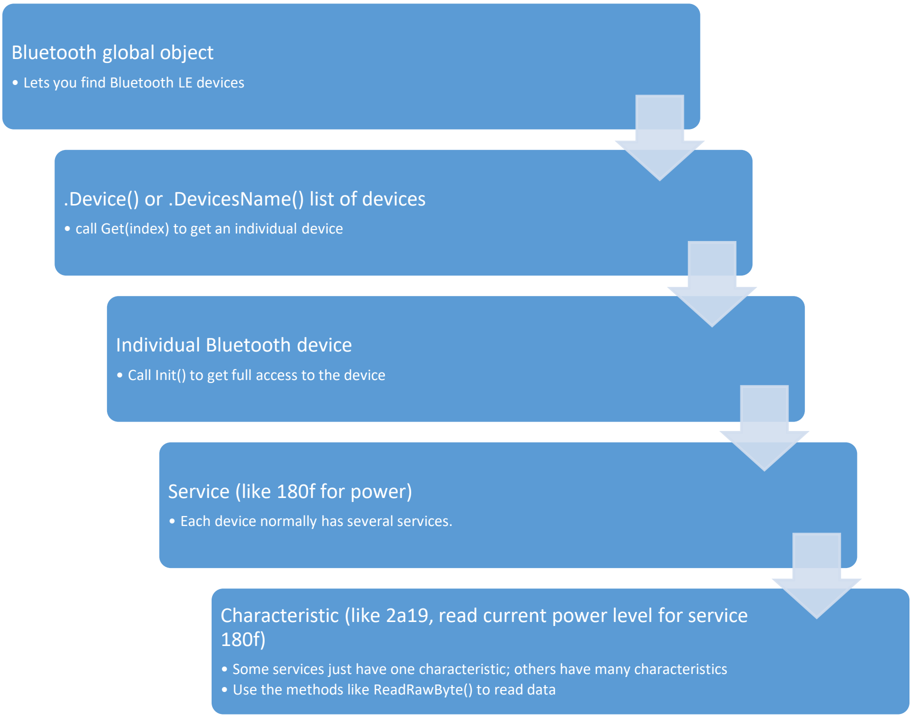
Bluetooth.Devices() returns a list of all the available Bluetooth devices regardless of their name.

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.

33.5 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., 0000**1800**-0000-1000-8000-00805f9b34fb). Each service in turn exposes a set of *characteristics* which are also specified with a short or long GUID.



Bluetooth global object

- Lets you find Bluetooth LE devices

.Device() or .DevicesName() list of devices

- call Get(index) to get an individual device

Individual Bluetooth device

- Call Init() to get full access to the device

Service (like 180f for power)

- Each device normally has several services.

Characteristic (like 2a19, read current power level for service 180f)

- Some services just have one characteristic; others have many characteristics
- Use the methods like ReadRawByte() to read data

The hierarchy of Bluetooth features.

33.5.1 Direct device Read routines

33.5.1.1 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.

33.5.1.2 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 ready 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”)
END
```

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 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)
END
```

33.6 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)[1]. method. The service and characteristic say which data value to set the notifications on. The value parameter is one of 0 = None,

[1] This exactly corresponds with the Windows Runtime WriteClientCharacteristicConfigurationDescriptorAsync method call. That name is much, much too long for a simple language like BC BASIC!

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
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
END
```

33.7 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
```

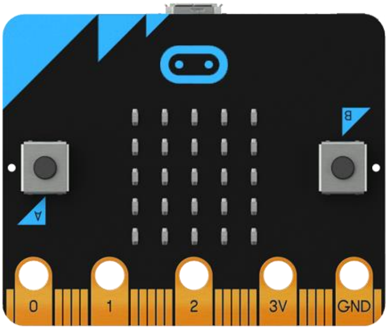
34 BLUETOOTH SPECIALIZATIONS FOR SPECIFIC DEVICES

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.

34.1 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 beLight; to get all of the devices call

devices = Bluetooth.DevicesName ("BBC micro:bit*").

To get the beLight specialization of a device, call tag = device.As("beLight"). .

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)

34.2 BELIGHT CC2540T LIGHT DEVELOPMENT 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
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.
SetColor (r, g, b, white)	Sets the color to a given red, green blue and white value.
ToString()	Prints out a little information about your device.

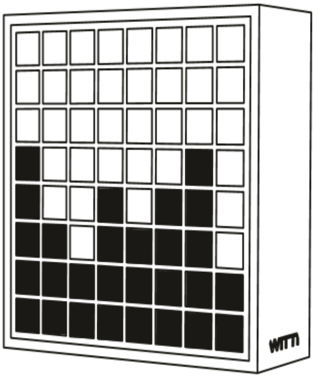
The device is mostly programmed through special service ffb0. The SetColor call is characteristic ffb5, and takes in 4 bytes for red, green, blue and white values.

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)

34.3 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 deviceis 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**
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.
GetPower()	Gets the current battery power of the device using service 180f characteristic 2a19. The value is not cached.
ChangeMode(mode)	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)	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. The Windows name of the device may not change until the device is reset and the re-paired.
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)

34.4 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**
GetName()	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached.
GetManufacturerName()	Gets the data from service 180a characteristic 2a29.
GetFirmwareRevision()	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.
GetPower()	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.

34.5 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*”)`.

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
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.
SetColor (r, g, b)	Sets the LED color to the given red, green and blue value. Columns are numbers 1 to 8.
SetOff()	Turns off the light
SetOn()	Turns on the light to the last color setting
ToString()	Prints out a little information about your device.

34.6 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 use 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 paring.

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 method; just look for the BT: Metawear Metamotion sample.

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.
GetPower()	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]])	When onoff is 1, will start to call function at the preferred rate. 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. The rate is in callbacks per second; the minimum value is .78125 and the maximum value is 3200. The callback function will be called with the Bluetooth specialization and the three values, x, y and z.
AltimeterSetup (onoff, fnc, speed)	When onoff is 1, set up the fnc called to be called when the altimeter data changes. The speed is in seconds between callback. Valid values are 4, 2, 1, 0.5, 0.25, 0.125 and 0.0625. 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. You can have either altimeter or barometer data, but not both.
BarometerSetup (onoff, fnc, [speed=1])	When onoff is 1, set up the fnc called to be called when the barometer data changes.

	The speed is in seconds between callback. Valid values are 4, 2, 1, 0.5, 0.25, 0.125 and 0.0625 The barometer callback will be called with the Bluetooth specialization and the the atmospheric pressure in pascals. You can have either altimeter or barometer data, but not both.
ButtonSetup (onoff, fnc)	When onoff is 1, set up the fnc function callback then the button is pressed. The callback will be called with the Bluetooth specialization and the button value (1=pressed and 0=not pressed)
GyroscopeSetup (onoff, function[, dps=500 [, rate=25]])	When onoff is 1, will start to call function at the preferred rate. The dps (degrees per second) is the precision of the gyroscope. Allowed values are 125, 250, 500, 1000, and 2000 The rate is in callbacks per second; the minimum value is 25 and the maximum is 3200 The callback function will be called with the Bluetooth specialization and the three values, x, y and z.
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)

fff0: D (fff3=D Data In): [writable], (fff5=C Command Channel)

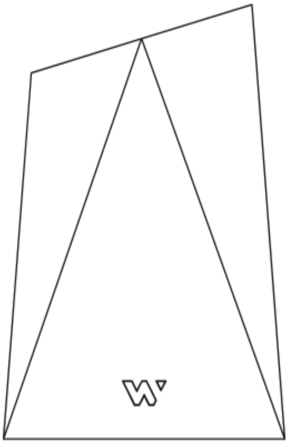
Example program: make a single reading of the temperature data

```
device = Bluetooth.PickDevicesName ("MetaWear")
IF (device.IsError)
    CLS RED
    PRINT "No MetaWear device found"
    PRINT device
    EXIT
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 "Availble 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.

34.7 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
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.
GetPower()	Gets the current battery power of the device using service 180f characteristic 2a19. The value is not cached.
AlarmSetting (type, r, g, b, advance)	Alarm settings. See SetAlarmTime 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)	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. Unlike the other NOTTI commands, SetName uses characteristic fff5, not fff3. The Windows name of the device may not change until the device is reset and the re-paired.
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 fff0. 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)

fff0: D (fff3=D Data In): [writable], (fff5=C Command Channel)

34.8 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
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 1/100s of a second. The x, y and z values are in “g” values and range +/-2.
BarometerSetup(onoff, period, callback)	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, ambient, object)	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
END
```

There are a number of sample programs in BC BASIC provided to demonstrate using the TI SensorTag.

34.9 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.

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**
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.
GetPower()	Gets the current battery power of the device using service 180f characteristic 2a19. The value is not cached.
AccelerometerSetup(onoff, period, callback) callback (tag, ax, ay, az, mx, my, mz, rx, ry, rz)	Sets the accelerometer to on 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.

	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)	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. 1=RED on 2=GREEN on 4=BUZZER on 0=all off The 1350 includes only a red LED; the 2650 has both a red and green LED.
IRSetup (onoff, period, callback) callback (tag, ambient, object)	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.
OpticalSetup (onoff, period, callback) callback (tag, lux)	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 The callback returns the tag and the light level in lux.
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 2=Gyro Y axis 4=Gyro X axis 7=Gyro all axis	8=Accel. X axis 16=Accel Y axis 32=Accel Z axis 56=Accel all axis	64=Magnetometer (all) (the Magnetometer does not allow you to turn on just one 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"
END
```

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 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.

35.1 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.

35.1.1 Set up the flow at Microsoft Flow

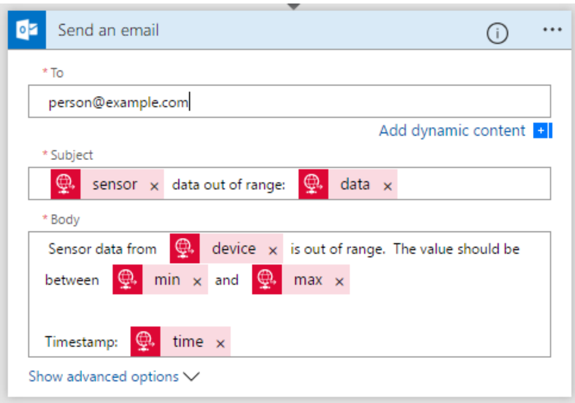
Your first step is to decide only your data format; you will need to provide your data schema when you create the flow. The data schema can be generated at http://jsonschema.net . In this example I picked a simple data-logging schema; example data and the resulting schema look like this

Example JSON data	Resulting Schema

{
"data": 82.3,
"time": "2017-03-26 3:23:45.2",
"device": "outside #2",
"sensor": "temperature",
"min": 70,
"max": 80
}
	{"$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"
	}

To create a flow at 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.



* To
person@example.com

* Subject
sensor × data out of range: data ×

* Body
Sensor data from device × is out of range. The value should be between min × and max ×

Timestamp: time ×

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.

35.1.2 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
END

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
```

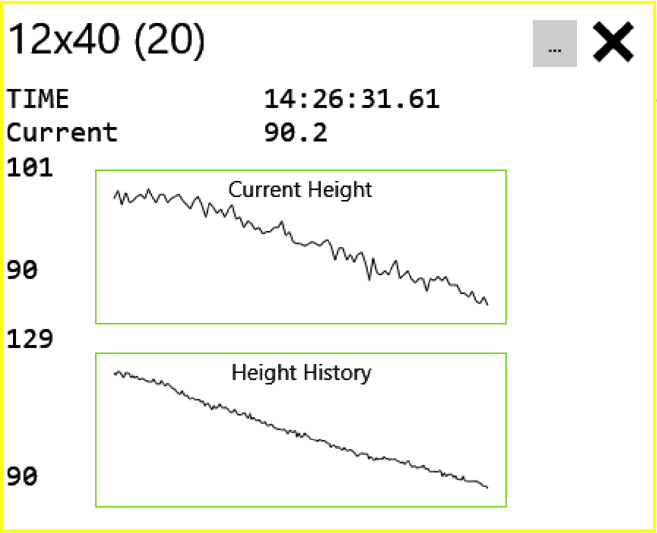
35.2 Hiking with an altimeter

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.

35.2.1 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.



```
TIME                 14:26:31.61
Current              90.2
```

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 remove (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.RequestRrlease()

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)

END

FUNCTION Button(meta, value)
    GLOBAL exitRequested
    IF (value = 1) THEN exitRequested = 1
END
```

35.2.2 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)
END

FUNCTION Button(meta, value)
   GLOBAL exitRequested
   IF (value = 1) THEN exitRequested = 1
END
```

35.2.3 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
END
```

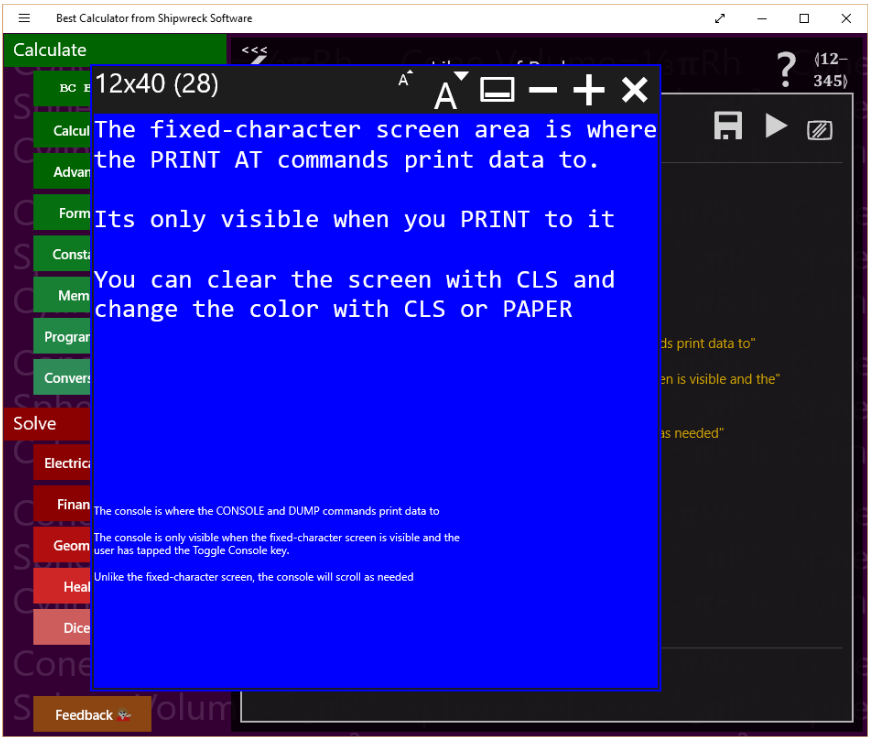
36 Graphics and BC BASIC

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.



```
The fixed-character screen area is where
the PRINT AT commands print data to.

Its only visible when you PRINT to it

You can clear the screen with CLS and
change the color with CLS or PAPER
```

The console is where the CONSOLE and DUMP commands print data to

The console is only visible when the fixed-character screen is visible and the user has tapped the Toggle Console key.

Unlike the fixed-character screen, the console will scroll as needed

Normally the console is not visible.

36.1 Fixed Character Screen Commands

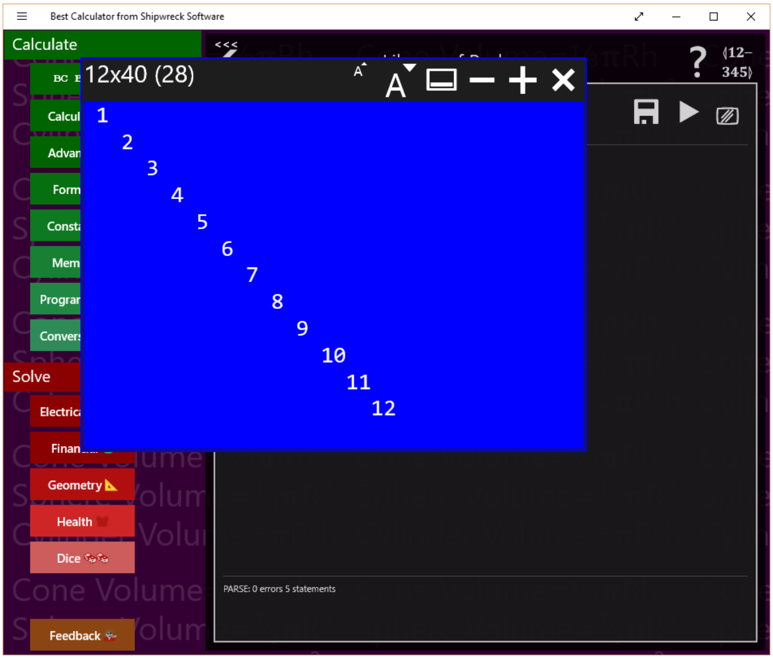
PRINT [AT line, column] <expression>

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:



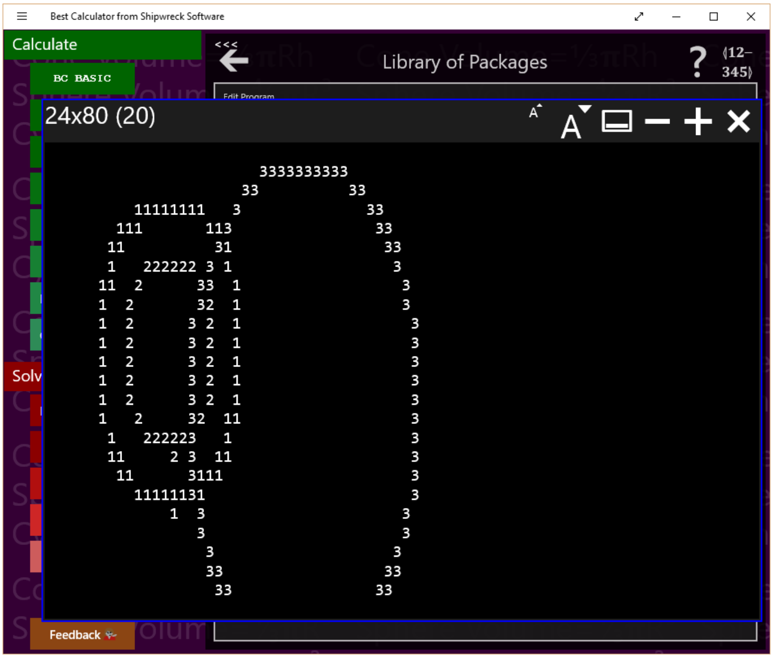
```
 1
   2
     3
       4
         5
           6
             7
               8
                 9
                   10
                     11
                       12
```

PARSE: 0 errors 5 statements

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
```

The circle program makes this output



```
                  3333333333
                33          33
    11111111   3              33
  111       113                33
 11          31                 33
 1   222222 3 1                  3
11  2      33  1                  3
1  2       32  1                  3
1  2      3 2  1                   3
1  2      3 2  1                   3
1  2      3 2  1                   3
1  2      3 2  1                   3
1  2      3 2  1                   3
1   2     32  11                   3
 1   222223   1                    3
 11     2 3  11                    3
  11      3111                     3
    11111131                       3
        1  3                      3
           3                      3
            3                    3
            33                  33
             33                33
```

36.2 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 possible change the color).

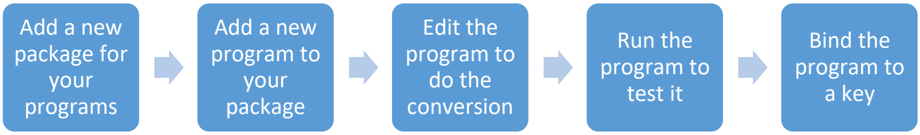
Example of using CLS to clear the screen and change the color:

```
CLS YELLOW
```

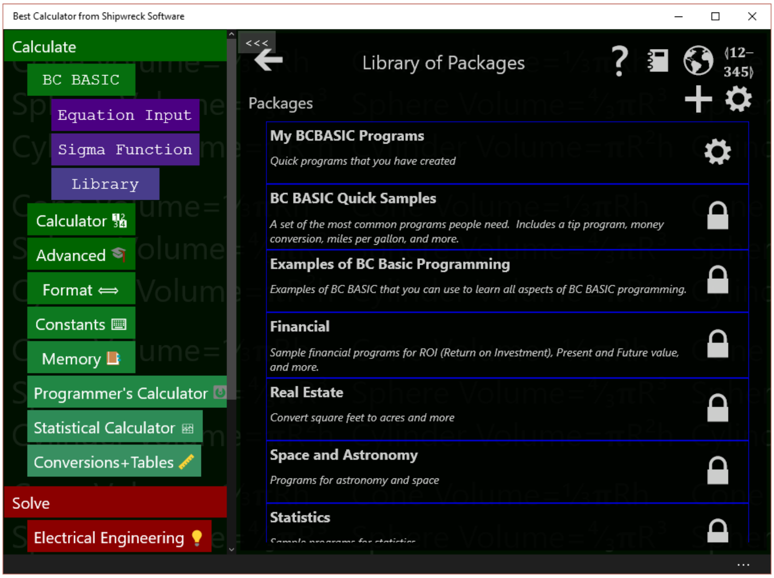
37 Using the Library, Step by Step

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.


Add a new package for your programs
Add a new program to your package
Edit the program to do the conversion
Run the program to test it
Bind the program to a key

37.1 ADD A NEW PACKAGE FOR YOUR PROGRAM



My BCBASIC Programs
Quick programs that you have created

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

Examples of BC Basic Programming
Examples of BC BASIC that you can use to learn all aspects of BC BASIC programming.

Financial
Sample financial programs for ROI (Return on Investment), Present and Future value, and more.

Real Estate
Convert square feet to acres and more

Space and Astronomy
Programs for astronomy and space

Statistics
Sample programs for statistics

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.

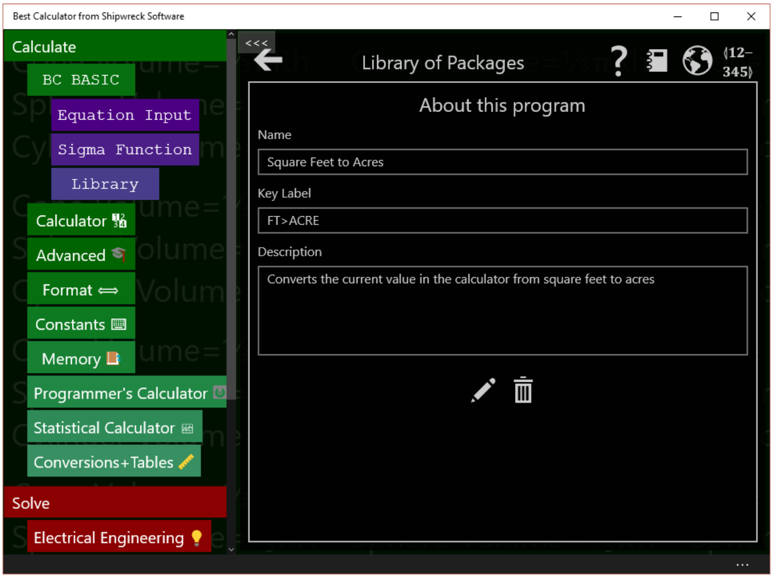
Tap the + key to add a new package. A new package will be created with a default name of "New Package".

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.

37.2 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.



About this program

Name

Square Feet to Acres

Key Label

FT>ACRE

Description

Converts the current value in the calculator from square feet to acres

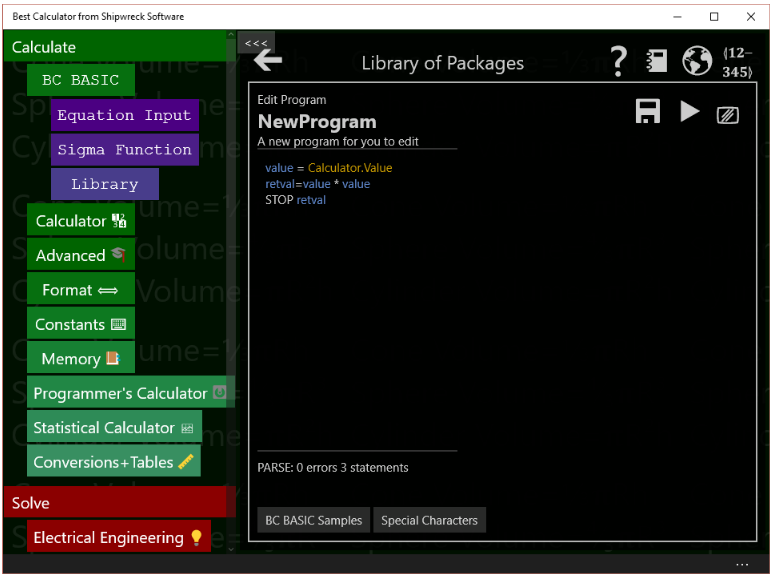
Change the name to "Square Feet to Acres" and the description to "Converts the current value in the calculator from square feet to acres". You don't have to do anything special to save the name and description; they are saved automatically when your program is saved.

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.

37.3 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.



Edit Program

NewProgram

A new program for you to edit

```
value = Calculator.Value
retval=value * value
STOP retval
```

PARSE: 0 errors 3 statements

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:

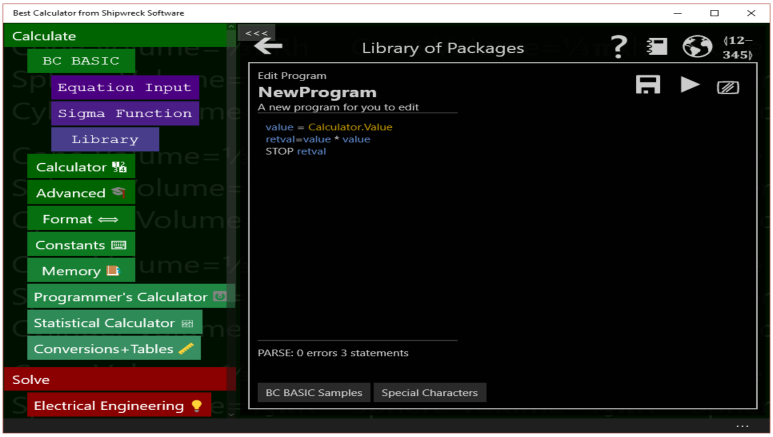
```
value = Calculator.Value
newValue = value / 43560
Calculator.Message = “Converted ”+ value ↲
      + “ square feet to acres”
STOP newValue
```

Each line corresponds to one of our steps. Enter this code into the program area.

37.4 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.



Edit Program
NewProgram
A new program for you to edit

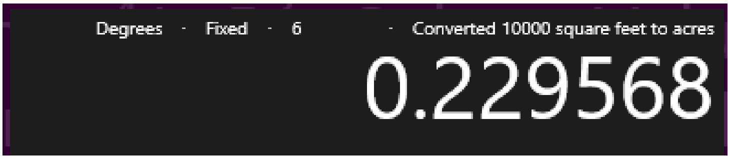
```
value = Calculator.Value
retval=value * value
STOP retval
```

PARSE: 0 errors 3 statements

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.



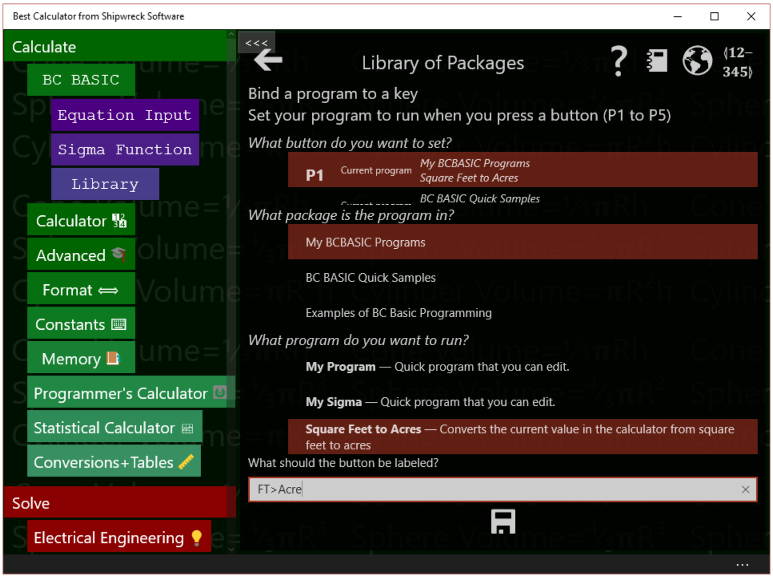
```
Degrees · Fixed · 6        · Converted 10000 square feet to acres
                                                       0.229568
```

37.5 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 (⟨12–345⟩); 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.



Library of Packages

Bind a program to a key

Set your program to run when you press a button (P1 to P5)

What button do you want to set?

P1 Current program *My BCBASIC Programs* *Square Feet to Acres*

Current program *BC BASIC Quick Samples*

What package is the program in?

My BCBASIC Programs

BC BASIC Quick Samples

Examples of BC Basic Programming

What program do you want to run?

My Program — Quick program that you can edit.

My Sigma — Quick program that you can edit.

Square Feet to Acres — Converts the current value in the calculator from square feet to acres

What should the button be labeled?

FT>Acre

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).

37.6 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.

38 Appendix: 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 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 programed.

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.

39.1 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.

39.1.1 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"
```

39.1.2 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
```

39.1.3 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
END
```

39.2 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.

39.2.1 Accelerometer

Demonstrates the basics of the AccelometerSetup 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
END
```

39.2.2 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"
END
```

39.2.3 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
END
```

39.2.4 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
END
```

39.2.5 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
```

39.2.6 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
```

39.2.7 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
```

```
END
```

39.2.8 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.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
    text = INPUT DEFAULT "hello" PROMPT "Text to write"
    speed  = INPUT DEFAULT 100 PROMPT "Scroll speed"
   PRINT "status",  tag.Write (text, speed)
END IF
```

39.3 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.

39.3.1 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
```

39.3.2 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
```

39.4 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.

39.4.1 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
```

39.4.2 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
```

39.4.3 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 DOTTI devices: when you use the DOTTI 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
END
```

39.4.4 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
END

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
```

39.4.5 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 DOTTI 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)
END
```

39.4.6 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
```

39.4.7 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
```

39.4.8 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
```

39.4.9 SetPixel to random green dots

Displays random green dots on the Dotti using the dotti.SetPixel command.

```
CLS BLUE
PRINT "Write green dot onto DOTTI 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
```

39.4.10 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
```

39.4.11 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
```

39.5 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.

39.5.1 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
```

39.5.2 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
```

39.5.3 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"
```

39.5.4 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
```

39.5.5 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()
END

FUNCTION ShowPedometer(tag)
   Screen.ClearLine(4)
   Screen.ClearLine(5)
   PRINT AT 4,1 "Steps", tag.GetSteps()
   PRINT AT 5,1 "Calorie", tag.GetCalories()
```

```
END

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()

END
```

39.5.6 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
```

39.5.7 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
```

39.6 BT: MagicLight

Supports the MagicLight and Flux lights. These are Bluetooth-enabled light bulbs for home use.

39.6.1 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
```

39.6.2 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
```

39.6.3 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
```

39.6.4 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 = light.SetColor (255, 0, 0)
   PRINT “status”, Status
END IF
```

39.7 BT: MetaWear MetaMotion

Demonstrates using the MetaMotion device from MetaWear. Metawear sells a variety of small battery-powered sensors devices with a variety of sensors includes accelerometers, gyroscopes, temperature and humidity sensors and more.

39.7.1 _Basics

A very simple program to get you started with programming the mbientlab.com MetaWear MetaMotion device. The program will let you pick a device and then report information about the device's name, manufacturer and current power usage.

```
device = Bluetooth.PickDevicesName ("MetaWear")
IF (device.IsError)
    CLS RED
    PRINT "No MetaWear device found"
    PRINT device
    EXIT
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 "Availble Methods", meta.Methods
```

39.7.2 Accelerometer

Demonstrates the meta.AccelerometerSetup(1, "Accelerometer", gforce, rate) call. When new acceleration data is sent from the device, the "Accelerometer" function will be called with the device plus an X, Y and Z value of the acceleration. Also calls Bluetooth.PickDevicesName ("MetaWear") to pick a device to connect to.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
   PRINT "No device was picked"
   END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")

gforce = INPUT DEFAULT 2 PROMPT "Maximum G-Force to measure (e..g, 2)"
rate = INPUT DEFAULT 25 PROMPT "Callbacks per second (e.g., 25)"

meta.ButtonSetup(1,"Button")
meta.AccelerometerSetup(1, "Accelerometer", gforce, rate)

ExitRequested = 0
MAXTIME = 100
FOR time = 0 TO MAXTIME STEP 1
   PRINT AT 8, 1 "TIME", time
   PAUSE 50
   meta.SetColor (time*255/MAXTIME, 0, 255)
   IF (ExitRequested =1)  THEN time = MAXTIME
NEXT time

REM Turn it off
meta.AccelerometerSetup(0, "Accelerometer")

FUNCTION Accelerometer(ble, x, y, z)
   PRINT AT 11, 1 "X", x
   PRINT AT 12, 1 "Y", y
   PRINT AT 13, 1 "Z", z
END


FUNCTION Button (ble, value)
   PRINT AT 4, 1 "Button!", ble
   PRINT AT 10,1 "VALUE", value
   GLOBAL ExitRequested
   IF (value = 1) THEN ExitRequested = 1
END
```

39.7.3 Altimeter

Demonstrates the meta.AltimeterSetup(1, "Altimeter", rate) call. When new altimeter data is sent from the device, the "Altimeter" function will be called with the device plus a height in meters. Also calls Bluetooth.PickDevicesName ("MetaWear") to pick a device to connect to.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
   PRINT "No device was picked"
   END
END IF
PRINT "Device is", device

tare = Math.NaN
meta = device.As ("MetaMotion")

rate = INPUT DEFAULT 4 PROMPT "Seconds between callbacks (e.g., 4)"

meta.ButtonSetup(1,"Button")
meta.AltimeterSetup(1, "Altimeter", rate)

ExitRequested = 0
MAXTIME = 100
FOR time = 0 TO MAXTIME STEP 1
   PRINT AT 7, 1 "TIME", time
   PAUSE 50
   meta.SetColor (time*255/MAXTIME, 0, 255)
   IF (ExitRequested =1)  THEN time = MAXTIME
```

```
NEXT time

meta.AltimeterSetup(0, "Altimeter")

FUNCTION Altimeter(ble, meters)
   GLOBAL tare
   IF (Math.IsNaN (tare)) THEN tare = meters

   REM Conversion factor is pasted from Bing
   feet=  meters * 3.2808399
   deltaFeet = (meters – tare) * 3.2808399
   PRINT AT 8, 1 "Tare", tare
   PRINT AT 9, 1 "Meters", meters
   PRINT AT 10, 1 "Feet", meters
   PRINT AT 11, 1 "Delta", deltaFeet

END



FUNCTION Button (ble, value)
   PRINT AT 4, 1 "Button!", ble
   PRINT AT 10,1 "VALUE", value
   GLOBAL ExitRequested
   IF (value = 1) THEN ExitRequested = 1
END
```

39.7.4 Ambient Light Sensor

Demonstrates the meta.LightSensorSetup(1, "LightSensor") call. When new ambient light data is sent from the device, the "LightSensor" function will be called with the device plus a light value. Also calls Bluetooth.PickDevicesName ("MetaWear") to pick a device to connect to.

```
CLS BLUE
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
```

```
   PRINT "No device was picked"
   END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")

meta.ButtonSetup(1,"Button")
meta.LightSensorSetup(1, "LightSensor")

ExitRequested = 0
MAXTIME = 100
FOR time = 0 TO MAXTIME STEP 1
   PRINT AT 8, 1 "TIME", time
   PAUSE 50
   meta.SetColor (time*255/MAXTIME, 0, 255)
   IF (ExitRequested =1)  THEN time = 9999
NEXT time

meta.AccelerometerSetup(0, "Accelerometer")

FUNCTION LightSensor(ble, lux)
   PRINT AT 11, 1 "LUX 1", lux
   REM PRINT AT 12, 1 "LUX 2", lux2
END



FUNCTION Button (ble, value)
   PRINT AT 4, 1 "Button!", ble
   PRINT AT 10,1 "VALUE", value
   GLOBAL ExitRequested
   IF (value = 1) THEN ExitRequested = 1
END
```

39.7.5 Barometer

Demonstrates the meta.BarometerSetup(1, "Barometer", rate) call. When new pressure data is sent from the device, the "Barometer" function will be called with the device plus a light value. Also calls

Bluetooth.PickDevicesName ("MetaWear") to pick a device to connect to.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
   PRINT "No device was picked"
   END
END IF
PRINT "Device is", device

meta = device.As ("MetaMotion")

rate = INPUT DEFAULT 4 PROMPT "Seconds between callbacks (e.g.,
4)"

meta.ButtonSetup(1,"Button")
meta.BarometerSetup(1, "Barometer", rate)

ExitRequested = 0
MAXTIME = 100
FOR time = 0 TO MAXTIME STEP 1
   PRINT AT 7, 1 "TIME", time
   PAUSE 50
   meta.SetColor (time*255/MAXTIME, 0, 255)
   IF (ExitRequested =1)  THEN time = MAXTIME
NEXT time

meta.AltimeterSetup(0, "Barometer")

FUNCTION Barometer(ble, pascal)
   REM Convert to inches of mercury
   inches = pascal * 0.000295299830714
   mb = pascal * 0.01
   PRINT AT 9, 1 "Pascal", pascal
   PRINT AT 10, 1 "Inches", inches
```

```
    PRINT AT 11, 1 "mb", mb

END



FUNCTION Button (ble, value)
    PRINT AT 4, 1 "Button!", ble
    PRINT AT 10,1 "VALUE", value
    GLOBAL ExitRequested
    IF (value = 1) THEN ExitRequested = 1
END
```

39.7.6 Black

Turns the LED to Black using the SetColor (red, green, blue) method.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
    PRINT "No device was picked"
    END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")
REM Color values are red, green, blue.
meta.SetColor(0, 0, 0)
```

39.7.7 Blue Pulse

Demonstrates the advanced capabilities of the LED pulse control in the MetaWear. The G, R, B values are set seperately; you set the high and low intensity and the rise, high, fall and total cycle time. This lets you design fancy pulse ability. This program sets the LED to slowly transition from a mid to high blue value and back again.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
```

```
IF device.IsError
    PRINT "No device was picked"
    END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")

REM meta.SetColor (0, 0, 0)
REM result = meta.LedConfig(0, 0, 0, 1000, 500, 20, 2500, 1)
REM result = meta.LedConfig(1, 0, 0, 1000, 500, 20, 2500, 1)
result = meta.LedConfig(2, 248, 160, 0, 100, 0, 1000, 0)
result = meta.LedOn()
PRINT result
```

39.7.8 Buttons

Demonstrates the meta.ButtonSetup(1, "Button") call. When teh user pressed the button, , the "Button" function will be called with the device plus an indication of the button press (0=up 1=down). Also calls Bluetooth.PickDevicesName ("MetaWear") to pick a device to connect to.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
    PRINT "No device was picked"
    END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")

meta.ButtonSetup(1,"Button")

MAXTIME = 100
FOR time = 0 TO MAXTIME
    PRINT AT 7, 1 "MAXTIME", MAXTIME
    PRINT AT 8, 1 "TIME", time
    PAUSE 50
    meta.SetColor (time*255/MAXTIME, 0, 255)
    IF (ExitRequested =1)
        time = MAXTIME
        ExitRequested = 0
    END IF
    IF(MAXTIME > 102)
        a = INPUT PROMPT "ERROR!"
    END IF
NEXT time

meta.ButtonSetup(0,"Button")

FUNCTION Button (ble, value)
    PRINT AT 4, 1 "Button!", ble
    PRINT AT 10,1 "VALUE", value
    GLOBAL ExitRequested
    IF (value = 1) THEN ExitRequested = 1
END
```

39.7.9 Green

Turns the LED to Green using the SetColor (red, green, blue) method.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
    PRINT "No device was picked"
    END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")
REM Color values are red, green, blue.
meta.SetColor(0, 255, 0)
```

39.7.10 Gyroscope

Demonstrates the meta.GyroscopeSetup(1, "Gyroscope", gforce, rate) call. When new gyroscope data is sent from the device, the "Gyroscope" function will be called with the device plus X, Y and Z values. Also calls Bluetooth.PickDevicesName ("MetaWear") to pick a device to connect to.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
    PRINT "No device was picked"
    END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")

dps = INPUT DEFAULT 500 PROMPT "Maximum Degrees-per-second
to measure (e..g, 500)"
rate = INPUT DEFAULT 25 PROMPT "Callbacks per second (e.g., 25)"

meta.ButtonSetup(1,"Button")
meta.GyroscopeSetup(1, "Gyroscope", dps, rate)

ExitRequested = 0
MAXTIME = 100
FOR time = 0 TO MAXTIME STEP 1
    PRINT AT 8, 1 "TIME", time
    PAUSE 50
    meta.SetColor (time*255/MAXTIME, 0, 255)
    IF (ExitRequested =1)  THEN time = MAXTIME
NEXT time

REM Turn it off
meta.GyroscopeSetup(0, "Gyroscope")

FUNCTION Gyroscope(ble, x, y, z)
```

```
    PRINT AT 11, 1 "X", x
    PRINT AT 12, 1 "Y", y
    PRINT AT 13, 1 "Z", z
END



FUNCTION Button (ble, value)
    PRINT AT 4, 1 "Button!", ble
    PRINT AT 10,1 "VALUE", value
    GLOBAL ExitRequested
    IF (value = 1) THEN ExitRequested = 1
END
```

39.7.11 LED Off

Turns the LED Off using the LedOff() method. This is different from simply setting the color to black.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
    PRINT "No device was picked"
    END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")
result = meta.LedOff()
PRINT result
```

39.7.12 LED On

Turns the LED on using the LedOn() method. It will display the last pattern set.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
```

```
    PRINT "No device was picked"
    END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")
result = meta.LedOn()
PRINT result
```

39.7.13 Magnetometer

Demonstrates the meta.MagnetometerSetup(1, "Magnetometer") call. When new magnetic data is sent from the device, the "Magnetometer" function will be called with the device plus an X, Y and Z value of the magnetic force. Also calls Bluetooth.PickDevicesName ("MetaWear") to pick a device to connect to.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
    PRINT "No device was picked"
    END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")

meta.ButtonSetup(1,"Button")
meta.MagnetometerSetup(1, "Magnetometer")

ExitRequested = 0
MAXTIME = 100
FOR time = 0 TO MAXTIME STEP 1
    PRINT AT 8, 1 "TIME", time
    PAUSE 50
    meta.SetColor (time*255/MAXTIME, 0, 255)
    IF (ExitRequested =1)  THEN time = MAXTIME
NEXT time
```

```
REM Turn it off
meta.MagnetometerSetup(0, "Magnetometer")

FUNCTION Magnetometer(ble, x, y, z)
    PRINT AT 11, 1 "X", x
    PRINT AT 12, 1 "Y", y
    PRINT AT 13, 1 "Z", z
END



FUNCTION Button (ble, value)
    PRINT AT 4, 1 "Button!", ble
    PRINT AT 10,1 "VALUE", value
    GLOBAL ExitRequested
    IF (value = 1) THEN ExitRequested = 1
END
```

39.7.14 Save Data To File

Saves Barometer data to a CSV file along with a time stamp. This sample demonstrates how you can make a simple data logger program that exports data in a format that can be used by Excel.

```
REM Version 4:52
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
    PRINT "No device was picked"
    END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")

file = File.AppendPicker("CSV file", ".csv", "altimeter.csv")

tare = Math.NaN
meta.ButtonSetup(1,"Button")
meta.AltimeterSetup(1, "Altimeter")
DataToWrite = ""
LastFlushTime = DateTime.GetNow()

10 REM LOOP TOP
ExitRequested = 0
MAXTIME = 10000
Screen.RequestActive()
FOR time = 0 TO MAXTIME STEP 1
   PRINT AT 8, 1 "TIME", time
   PAUSE 50
   meta.SetColor (time*255/MAXTIME, 0, 255)
   IF (ExitRequested =1)  THEN time = MAXTIME
NEXT time

IF (ExitRequested = 0) THEN GOTO 10

Screen.RequestRelease()
meta.AltimeterSetup(0, "Altimeter")

FUNCTION Altimeter(ble, type, meters)
   REM The first reading is the zero point
   GLOBAL tare
   IF (tare.IsNaN) THEN tare = meters
   PRINT AT 10,1 "TARE", tare, meters
   meters = meters – tare

   PRINT AT 11, 1 "TYPE", type
   PRINT AT 11+type, 1 "VALUE", meters

   GLOBAL file
   DIM line (3)
   dt = DateTime.GetNow()
   line(1) = dt.Date
   line(2) = dt.Time
   line(3) = meters
   str = String.Escape("csv", line)
```

```
    WriteData (str)
END


FUNCTION Button (ble, value)
    PRINT AT 4, 1 "Button!", ble
    PRINT AT 5,1 "VALUE", value
    GLOBAL ExitRequested
    IF (value = 1) THEN ExitRequested = 1
END

FUNCTION WriteData(line)
    GLOBAL DataToWrite
    GLOBAL LastFlushTime

    DataToWrite = DataToWrite + line

    Screen.ClearLine (16)
    Screen.ClearLine (17)
    Screen.ClearLine (18)
    Screen.ClearLine (19)

    now = DateTime.GetNow()
    PRINT AT 17,1 "LFT", LastFlushTime.Time, now.Time

    delta = now.Subtract (LastFlushTime)
    PRINT AT 16,1 "delta", delta

    IF (delta > 4)
        LastFlushTime = now
        FlushData()
    END IF
    PRINT AT 15, 1 "STR", str
END

FUNCTION FlushData()
    GLOBAL file
    GLOBAL DataToWrite
    result = file.AppendText (DataToWrite)
    DataToWrite = ""
    PRINT AT 18,1 "Write", result
END
```

39.7.15 Temperature

Demonstrates the meta.TemperatureSetup(1, "Temperature") call. When new temperature data is sent from the device, the "Temperature" function will be called with the device plus the temperature in degrees Celsius. Also calls Bluetooth.PickDevicesName ("MetaWear") to pick a device to connect to.

```
CLS
device = Bluetooth.PickDevicesName ("MetaWear")
IF device.IsError
    PRINT "No device was picked"
    END
END IF
PRINT "Device is", device
meta = device.As ("MetaMotion")

meta.ButtonSetup(1,"Button")
meta.TemperatureSetup(1, "Temperature")
ncall= 0

ExitRequested = 0
MAXTIME = 100000
FOR time = 0 TO MAXTIME STEP 1
    PRINT AT 8, 1 "TIME", time
    PAUSE 50
    IF (ExitRequested =1)
        time = MAXTIME
    ELSE
        meta.SetColor (time*255/MAXTIME, 0, 255)
        meta.TemperatureRead()
    END IF
NEXT time

meta.TemperatureSetup(0, "Temperature")



FUNCTION Temperature(ble, celcius)
    GLOBAL ncall
    ncall = ncall+1
    PRINT AT 11, 1 "TEMP", celcius
    PRINT AT 12, 1 "F.", (celcius*9/5+32)
    PRINT AT 13, 1 "NCALL.", ncall
END



FUNCTION Button (ble, value)
    PRINT AT 4, 1 "Button!", ble
    PRINT AT 10,1 "VALUE", value
    GLOBAL ExitRequested
    IF (value = 1) THEN ExitRequested = 1
END
```

39.8 BT: Notti

Demonstrates how to control the NOTTI device (from Witti design company). 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.

39.8.1 An introduction

An introduction to using the NOTTI specialization

```
CLS BLUE
PRINT "An introduction to the NOTTI specialization"

devices = Bluetooth.DevicesName (“*Notti”)
```

```
FOR i = 1 TO devices.Count
   device = devices.Get(i)
   PRINT "Notti Device"
   PRINT "Name", device.Name
   Notti = device.As (“NOTTI”)
   PRINT "Methods", Notti.Methods
   PRINT "GetName()", Notti.GetName()
   PRINT "GetPower()", Notti.GetPower()
   PRINT " "
NEXT i
```

39.8.2 Change Mode

Changes the mode of the NOTTI device (clock, animation, etc)

```
CLS BLUE
PRINT "Set NOTTI mode"
PRINT "0=Light on"
PRINT "1=Light off"
PRINT "2=Full animation"

device = Bluetooth.PickDevicesName(“*Notti”)
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   Notti = device.As (“NOTTI”)
   mode = INPUT DEFAULT 1 PROMPT "What mode?"
   Status = Notti.ChangeMode(mode)
   PRINT “status”, Status
END IF
```

39.8.3 NOTTI timer

Slowly switches from GREEN to RED over (n) minutes

```
device = Bluetooth.PickDevicesName(“*Dotti”)
```

```
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   notti = device.As ("NOTTI")
   PRINT "Power", notti.GetPower()

   nminutes  = Calculator.Value
   nminutes = INPUT DEFAULT nminutes PROMPT "How many minutes
to run for?"
END IF
STOP nminutes


FUNCTION DoTimer(device)
rstart = 0
rend = 255
gstart = 255
gend = 0
bstart = 0
bend = 0

notti.SetColor (rstart, gstart, bstart)
PRINT notti
Screen.RequestActive()
tot = nminutes*60
FOR s = 0 TO tot
   pct = s / tot
   r = pct*(rend - rstart) + rstart
   g = pct*(gend - gstart) + gstart
   b= pct*(bend - bstart) + bstart
   IF (Math.Mod(s, 60)= 0) THEN PRINT s/60
   Screen.ClearLine (8)
   PRINT AT 8,1 "Set color", tot-s,INT(pct*1000)/10
   notti.SetColor(r, g, b)
   PAUSE 50
NEXT s
PRINT "done!"
```

```
notti.SetColor(0, 0, 255)
Screen.RequestRelease()
RETURN
```

39.8.4 Raw Bluetooth commands

Sets the NOTTI color to red using the raw Bluetooth commands

```
CLS BLUE
PRINT "Set a NOTTI device to RED"

device = Bluetooth.PickDevicesName(“*Dotti”)
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    WriteColor(device, 255, 0, 0)
END IF

FUNCTION WriteColor(bt, r, g, b)
    bt.Init()
    REM The fff0 is the service for many NOTTI commands
    REM The fff3 is the characteristic used by service fff0
    REM     for many of the NOTTI commands
    REM the 6 and 1 are the bytes that define the NOTTI
    REM command to send (0x0601 means set LED color)
    REM the r g and b are the color to set.
    bt.WriteBytes (“fff0”, “fff3”, 6, 1, r, g, b)
END
```

39.8.5 Set Alarm

Sets the Alarm on the NOTTI device

```
CLS BLUE
PRINT "ALARM -- set NOTTI alarm"

device = Bluetooth.PickDevicesName(“*Dotti”)
```

```
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   Notti = device.As (“NOTTI”)
   PRINT "NOTTI", Notti
   h = INPUT DEFAULT 10 PROMPT "What hour?"
   m = INPUT DEFAULT 10 PROMPT "What minute?"
   ahead = INPUT DEFAULT 1 PROMPT "How far ahead 1=2.5 minutes"

   Status = Notti.SetAlarmTime(h, m)
   REM the 2 means it's a one-time alarm (0=off 1=every day)
   REM 255, 0, 0 is RED (rgb color)
   REM 1 means start up 2.5 minutes ahead of time (1=2.5 minute
10=25 minutes)

   PRINT "AT " + h + ":" + m + " turn to RED"
   PRINT "START " + (ahead*2.5) + " minutes ahead of that time"
   Status = Notti.AlarmSetting (2, 255, 0, 0, ahead)
   PRINT “status”, Status
END IF
```

39.8.6 SetColor to change the NOTTI color to blue

Sets the NOTTI color to Blue

```
CLS BLUE
PRINT "Sets a NOTTI to blue"

device = Bluetooth.PickDevicesName(“*Dotti”)
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   Notti = device.As (“NOTTI”)
   REM The three parameters are Red, Green and Blue values.
   REM They must be in the range 0 to 255
   Status = Notti.SetColor (0, 0, 255)
   PRINT “status”, Status
```

```
END IF
```

39.8.7 SetColor to change the NOTTI color to green

Sets the NOTTI color to Blue

```
CLS BLUE
PRINT "Sets a NOTTI to green"

device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   Notti = device.As ("NOTTI")
   REM The three parameters are Red, Green and Blue values.
   REM They must be in the range 0 to 255
   Status = Notti.SetColor (0, 255, 0)
   PRINT "status", Status
END IF
```

39.8.8 SetColorCustom animates the color from red to blue and back again

Uses the SetColorCustom command to change colors

```
CLS BLUE
PRINT "Sets a NOTTI from RED to BLUE and back again"

device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   Notti = device.As ("NOTTI")
   REM The three parameters are Red, Green and Blue values.
   REM They must be in the range 0 to 255
   Status = Notti.SetColorCustom (255, 0, 0, 0 , 0, 255)
   PRINT "status", Status
```

```
END IF
```

39.8.9 SetName of a NOTTI device

Sets the name of a NOTTI device. The device must be reset and re-paired for Windows to use the new name.

```
CLS BLUE
PRINT "Change the name of a NOTTI device"

device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   PRINT device.Name
   Notti = device.As ("NOTTI")
   Status = Notti.SetName ("My-Notti")
   PRINT "status", Status
END IF
```

39.8.10 Sync Time

Sets the time on the NOTTI device

```
CLS BLUE
PRINT "SyncTime -- set NOTTI time"

device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   Notti = device.As ("NOTTI")
   PRINT "NOTTI", Notti
   h = INPUT DEFAULT 10 PROMPT "What hour?"
   m = INPUT DEFAULT 10 PROMPT "What minute?"
   s = INPUT DEFAULT 15 PROMPT "What second?"
   Status = Notti.SyncTime(h, m, s)
   PRINT “status”, Status
END IF
```

39.9 BT: SensorTag 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).

39.9.1 Accelerometer Gyroscope and Magnetometer

Demonstrates the basics of the AccelometerSetup 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, rz, 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)

END
```

39.9.2 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"
```

39.9.3 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)
END

FUNCTION CTOF(C)
    F = C * 9/5 + 32
    F = Math.Round(F, 1)
END F

FUNCTION HPATOINCHM(HPA)
   ATM = HPA / 1013.25
   INCHM = ATM * 29.9213
   INCHM  = Math.Round(INCHM, 2)
END INCHM
```

39.9.4 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"
END
```

39.9.5 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)
END

FUNCTION CTOF(C)
    F = C * 9/5 + 32
    F = Math.Round(F, 1)
END F
```

39.9.6 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.PickDevicesName(“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
```

39.9.7 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)
END

FUNCTION CTOF(C)
    F = C * 9/5 + 32
    F = Math.Round(F*10) / 10
END F
```

39.9.8 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
END
```

39.10 BT: SensorTag 2541

Demonstrates how to use the TI SensorTag 2541 (the 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.

39.10.1 Accelerometer

Demonstrates the basics of the AccelometerSetup and using a callback routine.

```
CLS BLUE
PRINT AT 5,1 "Demonstrate TI SensorTag Accelerometer"

device = Bluetooth.PickDevicesName(“SensorTag”)
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As(“SensorTag2541”)
    PRINT AT 6,1 "Got a device", device.Name
    REM 1=turn on the on-device accelerometer
    REM 20=accelerometer update speed (in milliseconds)
```

```
   PRINT AT 7,1 "SETUP", tag.AccelerometerSetup(1, 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(3)
      PRINT "TIME", 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, x, y, z)
   Screen.ClearLine(1)
   PRINT x, y, z
END
```

39.10.2 Accelerometer Off

Turns off the accelerometer

```
CLS BLUE

device = Bluetooth.PickDevicesName(“SensorTag”)
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   tag = device.As(“SensorTag2541”)
   PRINT "Got a device", device.Name
   tag.SetupAcc(0, 100, “Acc”)
END IF
PRINT "All done"
```

39.10.3 Accelerometer to Magic Light

Uses the TI SensorTag accelerometer to drive a Magic Light

```
CLS BLUE
PRINT "Demonstrate TI SensorTag Accelerometer"

lights = Bluetooth.DevicesName ("LEDBlue*")
FOR i=1 TO lights
   device = lights.Get(i)
   light = device.As("MagicLight")
NEXT i

device = Bluetooth.PickDevicesName(“SensorTag”)
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   tag = device.As(“SensorTag2541”)
   PRINT "Got a device", device.Name
   PRINT tag.SetupAcc(1, 20, “Acc”)

   PRINT "Done with setup"
   FOR time = 1 TO 20
      PRINT AT 10, 1 time
      PAUSE 50
   NEXT time
   PRINT tag.SetupAcc(0, 0, "")
   tag.Close()
END IF
PRINT "DONE"

REM The tag.SetupAcc(1, 20, "Acc") call tells the device to call the "Acc"
REM function when the accelerometer changes.
FUNCTION Acc(tag, x, y, z)
   REM Tell the Acc function that the "light" variable is really a
   REM global variable defined in the main program.
```

```
    GLOBAL light

    Screen.ClearLine(1)
    PRINT x, y, z

    r = ABS (x*120)
    g = ABS(y*120)
    b = ABS (z*120)

    Screen.ClearLine(2)
    PRINT r, g, b

    light.SetColor (r, g, b)
END
```

39.10.4 Barometer

Demonstrates the basics of the BarometerSetup and using a callback routine.

```
devices = Bluetooth.DevicesName (“SensorTag”)

CLS BLUE
PRINT AT 5,1 "Demonstrate SensorTag Barometer measurements"

device = Bluetooth.PickDevicesName(“SensorTag”)
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As(“SensorTag2541”)
    REM 100=1000ms=1 second
    PRINT AT 7,1 "SETUP", tag.BarometerSetup(1,  100, “Barometer”)

    FOR time = 1 TO 30
        PAUSE 50
        PRINT AT 3,1 "TIME", 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(1)
    PRINT temp, pressure

    Screen.ClearLine(2)
    PRINT CTOF(temp), HPATOINCHM(pressure)
END

FUNCTION CTOF(C)
    F = C * 9/5 + 32
    F = Math.Round(F*10) / 10
END F

FUNCTION HPATOINCHM(HPA)
   ATM = HPA / 1013.25
   INCHM = ATM * 29.9213
   REM INCHM = Math.Round(INCHM*10) / 10
END INCHM

FUNCTION ROUND(val)
    val = Math.Round(val*10) / 10
END val
```

39.10.5 Button

Demonstrates the "Simple Key Service" on the SensorTag

```
devices = Bluetooth.DevicesName (“SensorTag”)

CLS BLUE
PRINT AT 5,1 "Demonstrate SensorTag Buttons"
PRINT AT 6,1 "Count", devices.Count

device = Bluetooth.PickDevicesName(“SensorTag”)
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As(“SensorTag2541”)
    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"
END
```

39.10.6 Gyroscope

Demonstrates the basics of the GyroscopeSetup and using a callback routine.

```
CLS BLUE
PRINT AT 5,1 "Demonstrate TI SensorTag Gyroscope"

devices = Bluetooth.DevicesName (“SensorTag”)

device = Bluetooth.PickDevicesName(“SensorTag”)
IF (device.IsError)
    PRINT "No device was picked"
```

```
ELSE
    tag = device.As(“SensorTag2541”)
    PRINT AT 6,1 "Got a device", device.Name
    REM 7=turn on all axis of the gyroscope
    REM 20=accelerometer update speed (in milliseconds)
    PRINT AT 7,1 "SETUP", tag.GyroscopeSetup(7, 20, “Gyroscope”)

    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
        PRINT AT 3, 1 "TIME", time
        PAUSE 50
    NEXT time
    REM Undo the accelerometer
    status = tag.GyroscopeSetup(0, 20, "Gyroscope")
    PRINT  AT 9, 1 "FINISH", status
    tag.Close()
END IF

FUNCTION Gyroscope(tag, x, y, z)
    Screen.ClearLine(1)
    PRINT ROUND(x), ROUND(y), ROUND(z)
END

FUNCTION ROUND(val)
    val = Math.Round(val*10) / 10
END val
```

39.10.7 Humidity

Demonstrates the basics of the HumiditySetup and using a callback routine.

```
devices = Bluetooth.DevicesName (“SensorTag”)

```

```
CLS BLUE
PRINT AT 5,1 "Demonstrate SensorTag Humidity measurements"

FOR i=1 TO devices.Count
   device = devices.Get(i)
    tag = device.As(“SensorTag2541”)
    REM 100=1000ms=1 second
    PRINT AT 7,1 "SETUP", tag.HumiditySetup(1,  100, “Humidity”)

    FOR time = 1 TO 30
        PAUSE 50
        PRINT AT 3,1 "TIME", 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(1)
    PRINT temp, humidity

    Screen.ClearLine(2)
    PRINT CTOF(temp), ROUND(humidity)
END

FUNCTION CTOF(C)
    F = C * 9/5 + 32
    F = Math.Round(F*10) / 10
END F

FUNCTION ROUND(val)
    val = Math.Round(val*10) / 10
END val
```

39.10.8 IR

Demonstrates the basics of the IRSetup and using a callback routine.

```
devices = Bluetooth.DevicesName (“SensorTag”)

CLS BLUE
PRINT AT 5,1 "Demonstrate SensorTag IR measurements"
PRINT AT 6,1 "Count", devices.Count

device = Bluetooth.PickDevicesName(“SensorTag”)
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As(“SensorTag2541”)
    REM 100=1000ms=1 second
    PRINT AT 7,1 "SETUP", tag.IRSetup(1,  100, “IR”)

    FOR time = 1 TO 30
        PAUSE 50
        PRINT AT 3,1 "TIME", 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(1)
    PRINT objTemp, ambTemp

    Screen.ClearLine(2)
    PRINT CTOF(objTemp), CTOF(ambTemp)
END

FUNCTION CTOF(C)
    F = C * 9/5 + 32
    F = Math.Round(F*10) / 10
END F
```

39.10.9 Magnetometer

Demonstrates the basics of the MagnetometerSetup and using a callback routine.

```
devices = Bluetooth.DevicesName (“SensorTag”)

CLS BLUE
PRINT AT 5,1 "Demonstrate SensorTag Magnetometer measurements"

device = Bluetooth.PickDevicesName(“SensorTag”)
IF (device.IsError)
   PRINT "No device was picked"
ELSE
   tag = device.As(“SensorTag2541”)
   REM 100=1000ms=1 second
   PRINT AT 7,1 "SETUP", tag.MagnetometerSetup(1,  100,
“Magnetometer”)

   FOR time = 1 TO 30
      PAUSE 50
      PRINT AT 3,1 "TIME", time
   NEXT time

   PRINT AT 8,1 "CLOSE", tag.MagnetometerSetup(0, 100,
"Magnetometer")
END IF

FUNCTION Magnetometer(tag, x, y, z)
   Screen.ClearLine(1)
   PRINT ROUND(x), ROUND(y), ROUND(z)
END



FUNCTION ROUND(val)
   val = Math.Round(val*10) / 10
END val
```

39.10.10 Raw access to SensorTag

Shows how to access the SensorTag using just the low-level Bluetooth functions and without using the specialization

```
CLS BLUE
devices = Bluetooth.DevicesName("SensorTag*")
ACCSERVICE ="f000aa10-0451-4000-b000-000000000000"
ACCCONFIG ="f000aa12-0451-4000-b000-000000000000"
ACCDATA ="f000aa11-0451-4000-b000-000000000000"



PRINT "COUNT", devices.Count

FOR i=1 TO devices.Count
   device = devices.Get(i)
   PRINT "DEVICE", device.Name

   address= device.Init()
   PRINT "Address", address

   REM Tell the device to send me some acc. data
   PRINT "ACC ON", device.WriteBytes(ACCSERVICE, ACCCONFIG, 1)

   PRINT "COUNT", "X", "Y", "Z"
   FOR time = 1 TO 15
      PAUSE 50
      data = device.ReadRawBytes(ACCSERVICE, ACCDATA)
      PRINT data.Count, data.Get(1), data.Get(2), data.Get(3)
   NEXT time

   REM Turn it back off
   PRINT "ACC OFF", device.WriteBytes(ACCSERVICE, ACCCONFIG, 0)
   PRINT " "
NEXT i
```

39.10.11 Raw callback with the SensorTag

A new program for you to edit

```
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
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 Accelometer
REM

FOR time = 1 TO 10
   PAUSE 50
   PRINT AT 1,1 time
   PAUSE 50
   PRINT AT 1,1 "     (clear)"
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)
   Screen.ClearLine(3)
   PRINT x, y, y
END
```

39.11 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.

39.11.1 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)
lmargin = 1+INT (( Screen.W – LEN str) / 2)
IF (lmargin < 1) THEN lmargin = 1
PRINT AT row,lmargin str
RETURN

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
```

39.11.2 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
```

39.11.3 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
```

39.11.4 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
```

39.11.5 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
```

39.11.6 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"
```

39.12 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.

39.12.1 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))
```

39.12.2 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
```

39.12.3 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
END

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
```

39.12.4 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]
```

39.12.5 Reading a CSV file

Ues 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
```

39.12.6 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
file.WriteText (String.Escape(“csv”, data))
```

39.13 EX: FINANCIAL

Sample financial programs for ROI (Return on Investment), Present and Future value, and more.

39.13.1 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
```

39.13.2 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
```

39.13.3 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 montly 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
```

39.13.4 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
```

39.13.5 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 rougly 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
```

39.13.6 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
```

39.13.7 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 intial investment of " +  INITIAL
STOP ROI
```

39.14 EX: Real Estate

Convert square feet to acres and more

39.14.1 Acres to square feet

Converts acres from from the calculator display into square feet

```
value = Calculator.Value
retval=value * 43560
Calculator.Message = "Converted " + value + " acres into square feet"
STOP retval
```

39.14.2 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
```

39.14.3 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 suply 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
```

39.14.4 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
```

39.14.5 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
```

39.15 EX: SPACE AND ASTRONOMY

Programs for astronomy and space

39.15.1 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
```

39.15.2 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
```

39.15.3 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
```

39.15.4 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
```

39.15.5 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
```

39.15.6 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
```

39.15.7 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
```

39.15.8 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 deltav = ve* ln (m0 / m1)
REM m0=initial mass
REM m1=final mass (after fuel is burnt)
REM ve = effective exhaust velocity (about 4400 for Space Shuttle main engines)

REM Solve the equation for m1 to return (m0–m1), the amount of propellant to burn
REM deltav = ve* ln (m0 / m1)
REM deltav / ve = ln(m0/m1)
REM exp(deltav / ve) = m0/m1
REM m0 / exp (deltav / ve) = m1
REM fuelburned = m0 – (m0 / exp(deltav/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
```

39.16 EX: STATISTICS

Sample programs for statistics

39.16.1 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
```

39.16.2 Margin of Error

Calculates the margin of error for samping 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 chosing .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
```

39.16.3 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
```

39.16.4 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 populate.  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
```

39.16.5 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), 93
- (subtract, BASIC), 92
- (subtract, Calculator), 21
!
  (factorial) key, 31
→M (store)
  In Calculator Memory Page, 39
→M (To Memory) key, 23
% (percent) key, 27
& (and) key, 44
( ) (parentheses, Calculator), 22
( ) (parenthesis)
  when calling a function, required, 91
( ) (parenthesis, BASIC)
  in expressions, 91
*(multiply), 93
** (raise to the power), 92
/ (divide), 93
[ ] (square brackets)
  in expressions, 91
^ (xor) key, 44
| (or) key, 44
~ (inverse) key, 44
+ (add), 93
+ (add, Calculator), 21
< (less than), 93
<= (less than or equal), 93
<> (not equal), 93
= (equal), 93
= (equals, Calculator), 21
> (greater than), 93
>= (greater than or equal), 93
± (change sign) key, 25
≪ (shift left) key, 46
≪+○ (rotate left) key, 46
≫ (shift right) key, 46
≫+○ (rotate right) key, 46
× (multiply, Calcualtor), 21
÷ (divide, Calculator), 21
√ (squarae root), 92
√ (square root) key, 25
∛ (cube root), 92
∛x key, 31
∜ (fourth root), 92
≅ (approximately equal), 93
⌫ (Delete) key, 22
◫ (word) key, 45
◫◫ (dword) key, 45
▯ (byte) key, 45
1/x (Inverse) key, 25
^{2} (square), 92
2's complement key, 44
2541, 189
^{3} (cube), 92
^{4} (fourth power), 92
∞ (infinity) key, 38
ABS (BASIC), 97
Abs key, 33
AccelerometerSetup, 189, 192
ACS (BASIC), 97
Add
  array, 110
AddRow
  array, 111
AlarmSetting, 186
Algebraic Entry, 21
Algebraic Entry System with Hierarchy, 22
Algebraic Operating System, 22
AND, 94
And (&) key, 44
AppendLine
  File, 128
AppendPicker

  File, 128
AppendText
  File, 128
array, 108
As
  Bluetooth, device.As(), 161
ASCII Table, 60
ASN (BASIC), 97
assignment, 119
AsTotalSeconds
  DateTime, 127
ATN (BASIC), 97
automatic graph, 147
Average
  Example BASIC program, 109
B# (count bits) key, 44
BarometerSetup, SensorTag 2541, 189, 193
beLight, 174
bin key, 43
Binary numbers, 43
bind a program to a key, 75
Bluetooth, 155
Bluetooth object, 159
Bluetooth.Devices, 160
Boxplots, 54
ButtonSetup, SensorTag 2541, 190, 193
byte (▯) key, 45
C (clear) key, 22
c (speed of light) key, 38
Calculator.Message, 125
Calculator.Value, 125
Calendar calculations, 41
CALL, 105
callbacks, Bluetooth, 166
Carriage return, 104
CC2540T, 174
CE (clear entry) key, 22
Ceil key, 33
Chain calculations, 21
ChangeMode, 175, 187
characteristic, Bluetooth, 163
CHR, 99
Circle
  Graphics, 145
Classical Statistics, 50
CLS, 106
  clearing the screen manually, 78
CODE, 99
Comment (using REM), 123
Console, 84
  CONSOLE command, 212
  CONSOLE Command, 107
  display or hide (toggle), 85
  DUMP command, 212
Constants (Calculator), 38
Conversions
  Between bases, 43
  BTUs, 57
  Bushels, 59
  Calories, 57
  Celsius, 58
  Centimeters, 58
  Cups, 59
  degrees to radians, 28
  Donuts, 57
  Ergs, 57
  Fahrenheit, 58
  Feet, 58
  Gallons, 59
  Grains, 58
  Grams, 58
  Inches, 57
  Joules, 57
  Kelvin, 58
  Kilograms, 58
  Kilometers, 58
  Kilowatt-Hours, 57

  Liters, 59
  Long tons, 58
  Maund, 58
  Meters, 58
  Miles, 58
  MMT, 58
  Ounces, 58
  Pecks, 59
  Pints, 59
  Pounds, 58
  Quarts, 59
  radians to degrees, 28
  Rankine, 58
  Sèr, 58
  Short tons, 58
  Therms, 57
  Tolä, 58
  Tonnes, 58
  Troy Ounces, 58
  Troy Pounds, 58
  Yards, 58
Correlation coefficient, 52
COS (BASIC), 97
Cos key, 28
Count, 50
  array, 110
count bits (B#) key, 44
csv
  String.Escape, 150
  String.Parse, 152
d→r key, 28
data boxes, 49
Date
  DateTime, 126
Date Calculations, 41
DateTime, 126
Day
  DateTime, 126
DayOfWeek
  DateTime, 126
dec key, 43
Decimal numbers, 43
degree and radians
  In Calculator, 28
degrees key, 28
degrees of freedom, 53
Desktop Shortcut, 62
df, 53
DIM, 108
Direct Algebraic Logic, 22
DOTTI, 175
double
  numbers stored as doubles, 87
DUMP, 111
dword (□□) key, 45
e key, 38
EE (scientific notation) key, 26
ELSE, 117
END, 123
Escape
  String, 150
Escape (stop program) key, 83
EXP (BASIC), 97
exponent, BASIC, 137
exponential notation (BASIC), 87
exponential notation (Calculator), 26
expressions, 90
F5 (run program) key, 83
Factorial (Calculator), 31
fff0, 177
fff0, NOTTI, 185, 188
fff3, 177
File, 128
First
  RemoveAlgorithm, 110
Fixed character size screen, 84
floats, 87

Floor key, 33
Flux light, 180
Font size
  indicator, 84
  larger, 84
  smaller, 84
FOR..NEXT loops, 112
Formatting (Calculator), 35
Frac key, 33
FUNCTION, 115
Get
  Bluetooth.Devices, 160
  Http, 131
GetAccelerometer, Hexiwear, 179
GetCalories, Hexiwear, 179
GetFirmwareRevision, Hexiwear, 178
GetGyroscope, Hexiwear, 179
GetHeart, Hexiwear, 179
GetHumidity, Hexiwear, 179
GetLight, Hexiwear, 179
GetMagnetometer, Hexiwear, 179
GetManufacturerName, Hexiwear, 178
GetMode, 179
GetNow
  DateTime, 126
GetPower, Hexiwear, 178
GetPressure, Hexiwear, 179
GetSteps, Hexiwear, 179
GetTemperature, Hexiwear, 179
GLOBAL, 115
g_n (gravitational constant) key, 38
GOSUB, 116
GOTO, 116
Graphics
  Screen, 145
GraphY
  Graphics, 147
Gregorian Calendar, 41
GUID, 163
GyroscopeSetup, SensorTag 2541, 190
Hebrew Calendar, 41
Herschel notation, 29
hex key, 43
Hex numbers, 43
Hexiwear, 178
Hijiri Calendar, 41
History of BASIC, 64
Hour
  DateTime, 126
Http, 131
HumiditySetup, SensorTag 2541, 190, 193
IF - THEN, 117
IMPORT FUNCTIONS, 118
import package, 74
infinite loop (stopping), 83
Init
  Bluetooth, 161
INPUT (operator, details), 96
INPUT (operator, preferred), 94
INPUT (statement, not preferred), 119
INT (BASIC), 97
Integer key, 33
Integers, 87
Intercept, 52
Inverse Cos (calculator), 29
Inverse Sin (calculator), 29
Inverse Tan (calculator), 29
IRSetup, SensorTag 2541, 190, 194
IsError, 90
IsNaN, 90

IsNumber, 90
Iso8601
  DateTime, 127
IsObject, 90
IsString, 90
json
  String.Escape, 152
  String.Parse, 153
Julian Calendar, 41
Keyboard calculator button, 63
LEDBlue, 180
LEFT, 98
LEN, 99
LET, 119
library
  delete packagae, 77
  export package, 77
  how to display properties, 73
  made up of packages, 72
libray
  add package, 214
Light, 174
Line
  Graphics, 145
Line ending
  \n, 86
  \r, 86
  \v, 86
Linear Regression, 52, 55
Linear Regression Chart, 52
LN (BASIC), 97
ln key, 30
LoadScreenFromMemory, 176
log key, 30
$\log_2$ key, 30
Logarithms (Calculator), 30
Looping (FOR..NEXT loops), 112
M- (memory subtract) key, 24
M→ (recall)
  In Calculator Memory Page, 39
M→ (Recall) key, 23
M-(memory subtract)
  Calculator Memory Page, 39
M+ (memory add)
  In Calculator Memory Page, 39
  key, 24
MagicLight, 180
MagnetometerSetup,
  SensorTag 2541, 191
Math.Abs, 134
Math.Acos, 134
Math.Asin, 134
Math.Atan, 134
Math.Atan2, 134
Math.Ceiling, 134
Math.Cos, 134
Math.Cosh, 134
Math.DtoR, 134
Math.E, 138
Math.Exp, 137
Math.Factorial, 138
Math.Floor, 134
Math.Frac, 135
Math.IsNaN, 138
Math.Log, 137
Math.Log10, 137
Math.Log2, 137
Math.Max, 135
Math.Min, 135
Math.Mod, 135
Math.NaN, 138
Math.PI, 138
Math.Pow, 137
Math.Round, 135
Math.RtoD, 134
Math.Sign, 136
Math.Sin, 134

Math.Sinh, 134
Math.Sqrt, 137
Math.Tan, 134
Math.Tanh, 134
Math.Truncate, 136
Max
  array, 110
MaxCount
  array, 110
MaxOf, 110
mbientlab.com, **181**
Mean, 50
median, 51
Memory
  M+ (memory add) key, 24
  Memory Recall (M→) key, 23
  Memory Store (→M) key, 23
  Memory Subtract (M-) key, 24
  Naming memory, 39
Memory Page, 23, 39, 41
Memory.<constant_name>, 140
Memory.GetOrDefault, 140
Memory.IsSet, 140
Memory[<expression>], 139
MetaMotion, **181**
MetaWear, **181**
MID, 98
mikroElektronika, 178
Min
  array, 110
MinOf, 110
Minute
  DateTime, 126
Mod (modulo) key, 31
Month
  DateTime, 126
N (count), 50
N_a (Avogaddro's number) key, 38
NaN key, 38
NEXT, 112
NOT, 94
NOTTI, 186
oct key, 43
Octal numbers, 43
OR, 94
Or (|) key, 44
Output screen, 84
  close, 85
  CLS, 106
  larger, 85
  PRINT, 121
  smaller, 85
P10 (10% percentile), 51
P90 (90% percentile), 51
package
  about, 76
  delete, 77
  how to add, 214
  how to display properties, 73
  how to export, 77
  how to import, 74
  made up of programs, 78
Parentheses (Calculator), 21
Parse
  String, 152
PAUSE, 120
Percent (%) key (%), 27
Percent Discount, 27
Percent Formatting (Calculator), 37
PI, 91
PickDevicesName, 161
Population Standard Deviation, 50
Post
  Http, 132
PRINT, 121
program

  about, 80
  add, 78
  delete, 81
  edit, 79
  edit dialog, 82
  edit from the About dialog, 80
  how to add, 215
  how to bind to a program key, 218
  how to run, 217
  properties, 79
  run, 79
  running from edit dialog, 82
  saving while editing, 82
program list
  how to display, 73
Programmer’s calculator, 43
Put
  Http, 132
p-value, 53
Q1 (1st quartile), 51
Q2 (2nd quartile), 51
Q3 (3rd quartile), 51
r→d key, 28
radians and degrees (Calculator), 28
radians key, 28
RAND (set random seed), 122
Random
  RemoveAlgorithm, 110
Random Numbers (BASIC), 122
Random Numbers (Calculator), 34
Read, Bluetooth device, 164
ReadAll
  File, 129
ReadCachedByte, 164
ReadCachedBytes, 165
ReadLines
  File, 129
ReadPicker
  File, 129
ReadRawByte, 164
ReadRawBytes, 165
Recall (M→), 23
Rect
  Graphics, 145
Regression, 52
Relative Standard Deviation, 50
REM (comment), 123
RemoveAlgorithm
  array, 110
RequestActive, 143
RequestRelease, 143
Reservoir Sampling
  array, 110
Rfc1123
  DateTime, 127
RIGHT, 98
RND, 91, 122
rnd key, 34
rnd N key, 34
Robust Statistics, 51
rotate left (≪+○) key, 46
rotate right (≫+○) key, 46
Round key, 33
Rounding (Calculator), 33
RSD, 50
Sales Tax, 27
Sample Standard Deviation, 50
save (export) package, 77
SaveScreenToMemory, 176
Scatterplots, 55
Scientific (exponential) notation (BASIC), 87
Scientific (exponential) Notation (Calculator), 26
screen clear key (BASIC), 78
Screen.ClearLine, 143

Screen.ClearLines, 143
Screen.H, 144
Screen.RequestActive, 143
Screen.RequestRelease, 143
Screen.W, 144
Second
  DateTime, 126
SensorTag 2541 (original), 189, 192
SensorTag*, 189, 192
service, Bluetooth, 163
SetAlarmTime, 187
SetColor, beLight, 174
SetColor, MagicLight, 180
SetColor, NOTTI, 184, 187
SetColorCustom, NOTTI, 187
SetColumn, 176
SetName, 176
SetName, NOTTI, 187
SetNameArbitrary, 176
SetNameArbitrary, NOTTI, 187
SetOff, MagicLight, 180
SetOn, MagicLight, 180
SetPanel, 176
SetPixel, 176
SetPosition
  Graphics, 146
SetRow, 176
SetSize
  Graphics, 146
SGN (BASIC), 98
shift left (≪) key, 46
shift right (≫) key, 46
SIN (BASIC), 97
Sin key, 28
Size
  File, 128, 130
Slope, 52
Smart quotes, 104
  and strings, 88
specializations, Bluetooth, 169
SQR (BASIC), 97
Standard Deviation, 50
statement, 101
statement terminator (not in BASIC), 86
Statistical calculator, 48
Statistics
  Entering data, 49
StdErr Line, 52
StdErr Slope, 52
STOP, 123
Store (→M) key, 23
String constants, 88
String Extension, 150
Student’s t-test, 53
Subtract
  DateTime, 126
Sum, 50
SWAB (swap bytes) key, 45
SyncTime, 176
SyncTime, NOTTI, 187
t statistic, 53
TAN (BASIC), 97
Tan key, 28
Texas Instruments, 189, 192
Texas Instruments, beLight CC2540T, 174
Time
  DateTime, 126
TimeHHmm
  DateTime, 126
trigonometry, 28
T-Tests, 53
Tukey boxplots, 48
Unicode minus signs, 88
Unicode table (BASIC), 104
Unicode table (Calculator), 61
Update
  Graphics, 148

VAL, 100
Variables, 88
  ending in $ (dollar sign), 89
Visually Perfect Algebraic Method, 22
Welch’s t-test, 53
Witti Design, 175, 186
word (◫) key, 45
WriteCallbackDescriptor, Bluetooth, 166
WriteClientCharacteristicConfigurationDescriptorAsync, 166
WriteLine
  File, 130
WritePicker
  File, 130
WriteText
  File, 130
$\bar{x}$ (mean), 50
x^2 (Square) key, 25
x^3 key, 31
Xor (^) key, 44
x^y key, 32
XY Scatterplots, 55
$y\sqrt{x}$ key, 32
Year
  DateTime, 126
π key, 38
s (sample standard deviation), 50
Σ (sum), 50
$\boldsymbol{\sigma}_n$ (population standard deviation), 50