IOT Additions to Best Calculator

## COMPLETE OTHER UPDATES

### COMPLETE: NAN

The NaN indexing is bad.  
Added Math.NaN because it’s already documented, sort-of and used by at least one program(!)

### COMPLETE Array

Explain how to append to an array  
Add an “Append”?

DIM list  
list(ist.Count+1) = …

### COMPLETE DateTime

Dt = DateTime.GetNow()  
Dt.Date  
Dt.Time  
Dt2 = DateTime.GetNow()  
Dt2.Subtract(Dt) 🡪 time in seconds (as a double)

Should also have Dt.TimeHM, TimeHMS

## NEXT VERSION! Graphics

BC Basic now includes powerful graphics abilities!

Simplest possible graphics:  
w = Graphics.Window(x,y,w,h) [all in pixels]  
w.Color = Colors.Blue [or just BLUE]  
w.Line (x1, y1, x2, y2)  
w.Text(x1, y1, “my text”)  
  
Next level: scaling  
REM make a graph with a central area  
w = Graphics.Window(x,y,w,h) [all in pixels]  
central = w.Window (x,y,w,h) [relative to first]  
central.Color = RED  
central.Scale(xmin, xmax, ymin, ymax)  
central.Line(x1, y1, x2, y2)  
xaxis = Graphics.Window(x, y, w, h)  
xaxis.Scale(xmin, xmax, 0, 1)  
xaxis.Text(x, 0.1, “My text”) [0.1=bottom of the area!]  
  
Push/Pop graphics state  
w.GPush()  
w.GPop()  
  
Rotate  
central.Center (x,y)  
central.Rotate(.3\*PI)  
  
3D!

## COMPLETE: PRINT with \r, \r\n and \n

PRINT now correctly handles strings with any combination of \r, \r\n and \n characters. Each of these is considered a newline and will start the output on the next line.

A common source of strings with embedded carriage control characters is data from the Internet retrieved with the Http.Get method.

## COPIED OVER: DONE: Connecting to Microsoft Flow

Microsoft Flow is a cloud-based utility that performs actions (like sending email) based on triggers. Best Calculator can trigger the *Request* type trigger using the Http.Post method.



Http is only available to Best Calculator, IOT Edition.

Microsoft has good documentation about Flow and the Request trigger

* [Calling Microsoft Flow](https://flow.microsoft.com/en-us/blog/call-flow-restapi/) from your application (Irina Gorbach)
* [Walkthrough](https://flow.microsoft.com/en-us/blog/slash-commands/): Using the Request trigger

As an FYI, “Web Hooks” is the inverse of the Request trigger. It’s not supported in BC Basic.

TODO: make a clearer POST request (post up the data in individual chunks + explanation and then have the Flow put the data together. And hide the URL in a named memory cell, and include a pop-up to get the correct url.

Example Program

temp = 62.3  
subject= String.Escape(“json”, “Alert: temperature=” + temp)  
body= "The temperature is too high. Temp="+temp  
json="{'emailSubject':”+subject+”, 'emailBody':”+body+”, 'emailaddress':'pesmith@microsoft.com'}"  
url= “<https://prod-28.westus.logic.azure.com:443/workflows/ec657eceb2364ce6898b79b83ca786a6/triggers/manual/run?api-version=2015-08-01-preview&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=sF-BGuF5oSgH2ctHtm8MqNITscfm-jdc4VCvdEw5weE>”  
result = Http.Post(url, json, “application/json”)  
PRINT result.code  
PRINT result.content

## COPIED IP: TODO: File Append: AppendPicker and …

## COMPLETE File Reading: ReadPicker and Read methods

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.

### myfile=File.ReadPicker(“.txt”)

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:

file = File.ReadPicker (“.json”)  
data = file.ReadAll()  
CLS GREEN  
PRINT "All the data as one big string"  
PRINT data  
  
file = File.ReadPicker (“.json”)  
lines =file.ReadLines()  
FOR i=1 TO lines.Count  
 PRINT lines(i)  
NEXT i

You can also provide a list of extensions that you are willing to accepts

DIM extensions()

extensions(1) = ".json"

extensions(2) = ".txt"

extensions(3) = ".csv"

CLS BLUE

PRINT extensions.Count

file = File.ReadPicker(extensions)

PRINT file

## COMPLETE: Http to GET, PUT and POST to the internet

HTTP is used by IOT to get information and commands from a central server and to put data up to a server. The BC Basic Http object give you a simple interface to GET, PUT and POST to these APIs.

Http is only available to Best Calculator, IOT Edition.

For example, ycombinator.com has a news feed; an example URL for their data is <https://hacker-news.firebaseio.com/v0/item/8863.json?print=pretty>

You can get this data with response = Http.Get (url). The actual result contents are available from the response.Result. The HTTP Status code (like 200 for success) is available from response.StatusCode . The response reason is available from response.ReasonPhrase.

You can parse the JSON data with String.Parse (“json”, data)

## COPIED: String object to escape and parse string

The String object lets you escape strings in different formats and parse data from strings.

### TODO: String.Escape (“csv”, <string or array>)

TODO: function is written. Can handle string or array.

### String.Escape (“json”, <string>)

For example, suppose you have a string and you want to put it into a larger JSON string. The string will need to be enclosed in quotes and some characters (like embedded quotes) will need to be escaped.

Value = String.Escape (<escape>, <string>) will escape the given string using the escape type presented.

The CSV escape can take either a single value or an array. When it takes an entire array, the returned string is a comma-separated string of all the values, ending with a CR LF. The CSV format used is from [RFC 4180](https://tools.ietf.org/html/rfc4180), extended to allow Unicode characters.

### TODO: String.Parse(“csv”, <string>)

The function is written and has an example in CSV Read

### String.Parse(“json”, <string>)

value = String.Parse (<parse>, <string>) will parse the input string as a value. The value might be a string, an array, and possibly an array of arrays.

For example, you might have read in a JSON object and you need to convert it to an array. Just call array = String.Parse (“json”, inputValue) to convert the inputValue string into an array. The resulting array might be a nested array (and array of strings and arrays)

Suppose the input string is

{

"by" : "dhouston",

"descendants" : 71,

"id" : 8863,

"kids" : [ 8952, 9224,],

"score" : 111,

"time" : 1175714200,

"title" : "My YC app: Dropbox - Throw away your USB drive",

"type" : "story",

"url" : "http://www.getdropbox.com/u/2/screencast.html"

}

You call this code: array = String.Parse (“json”, inputValue). Then you can pull out data like this:   
PRINT array.by

This statement will print dhouston.

data = “{ "by" : "dhouston", "id" : 8863, "kids" : [ 8952, 9224], "score" : 111 }”

CLS GREEN  
PRINT "All the data as one array"  
PRINT json  
PRINT json.by

The JSON parse does not maintain the order of the data. Pulling the data out by index (like PRINT json(1)) will not work reliably.

## front of BBC micro:bitIP: 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 Microbit 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 instead the device was running a program that just does Bluetooth.

A hex file that reprograms your BBC micro:bit to do just that is available at \_\_\_

A microbit uploader that will automatically move a hex file to your BBC micro:bit is available from [touchdevelop](https://www.touchdevelop.com/microbituploader).

<https://github.com/lancaster-university/microbit-docs/tree/master/docs/ble>

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 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. |
| SetColor (r, g, b, white) | Sets the given column to the given red, green and blue value. Columns are numbers 1 to 8. |
| ToString() | Prints out a little information about your DOTTI 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)

## COPIED OVER IP: 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 a number of 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 web site at <https://mbientlab.com/>

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

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. |
|  |  |
|  |  |
|  |  |
| SetColor (r, g, b) | Sets device color to the given red, green and blue 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)

## IP: Updating the C# code

An exciting feature of Best Calculator, IOT Edition is the ability to extend the features of the calculator. This example walks through the steps used to support the BBC microbit developer board.

The BBC [microbit](http://microbit.org/) is a small programmable board that has been given to all “Year 7” (11 and 12 year-old) students in Great Britain. It includes buttons, LEDs, accelerometers and more.

### Create the first function

The simplest function to add is the Write function which will cause the Microbit to display some text. The BC BASIC test code we want to call will look like this:

device = Bluetooth.PickDevicesName(“\*BBC\*”)  
bt = device.As(“microbit”)

### Add the device file

The new class file, BbcMicrobit.cs was added to the Bluetooth/Device folder. The code from the BeLightTi.cs file was copied in and the class name changed to BbcMicrobit.

This function simply involves updating characteristic E95D93EE251D470AA062FA1922DFA9A8 with the text to display. The RunAsync() method in the class is where functions are called; we’ll be adding some new stuff there.

The TI device only ever needed single bytes written; the BBC device will need to write a string. We will need a new function that, given a string, will write it to an arbitrary service and characteristic.

private async Task<RunResult.RunStatus>

CommonDeviceCallString(

string serviceString,

string characteristicString,

String arg,

BCValue Retval)

{

var encoding = new System.Text.UTF8Encoding();

var buffer = encoding.GetBytes(arg);

return await BluetoothDevice.DoWriteBytes(

Device, serviceString,

characteristicString, buffer, Retval);

}

We next add the actual function call which we will call “Write”. The function calls are all handled in the RunAs dispatch method. Add a little case statement to handle the Write call.

case "Write": // Writes a single string

if (!BCObjectUtilities.CheckArgs  
 (1, 1, name, ArgList, Retval))   
 return RunResult.RunStatus.ErrorStop;

var str = (await ArgList[0].EvalAsync(context)).AsString;

return await CommonDeviceCallString(

"E95DD91D251D470AA062FA1922DFA9A8",   
 "E95D93EE251D470AA062FA1922DFA9A8",   
 str, Retval);

The BC BASIC system needs to be told about the new type of device; this is done in BluetoothDevice.cs in the DoAs() method. This method handles the device.As() call that converts a generic Bluetooth device into a specific one like the BBC Microbit.

Add this line to the DoAs method switch (deviceName) statement:

case "microbit":   
Retval.AsObject = new Devices.BbcMicrobit(ble);   
return RunResult.RunStatus.OK;

Now we can give it a try!