### String.ToUpper(str) and String.ToLower(str)

The String.ToUpper(str) method returns the original string but converted to upper case. For example

str = String.ToUpper(“Apple”)  
PRINT str

will print “APPLE”. Similarly,

str = String.ToLower(“Apple”)  
PRINT str

will print “apple”. This is often used to convert user input into a more regular input for the computer to deal with.

## Working with Arduino

The Arduino is a very popular small microcontroller. Thanks to its simple but powerful IDE (integrated development environment), many libraries and low price, it’s become perhaps the most common hobbyist microcontroller. You can find more information at <https://www.arduino.cc/>

Many Arduino devices are connected via a Bluetooth Rfcomm (also known as SPP, Serial Port Protocol) mini-board. Many are available from different vendors.

To communicate with an Arduino using a Bluetooth serial interface, follow the instructions in Rfcomm (Serial-port) Bluetooth. There are methods like PickDevicesRfcommName (“\*”) to let your end-user pick a Bluetooth device to connect to and to send and receive data.

There isn’t a single Bluetooth protocol for these connections. Many devices (like the Slant Robotics LittleBot and Infineon DPS310) have their own mini-protocol. Other boards use “standard” protocols like the Ardudroid.

If there’s some protocol you would like supported, please contact [shipwrecksoftware@live.com](mailto:shipwrecksoftware@live.com) and let us know what protocol and what you’re building!

## C:\Users\toomr\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ArdudroidProtocol.pngArdudroid Protocol

The Ardudroid protocol is a simple Rfcomm protocol for controlling an Arduino device. It includes commands for reading and writing the different pins. The Ardudroid specialization is built on an Rfcomm device.

The protocol documentation is at <http://www.techbitar.com/ardudroid-simple-bluetooth-control-for-arduino-and-android.html> . The Ardudroid specialization also includes two command extensions: ServoAttach and ServoMove to controlling servo motors. You need to download the ardudroid.ino file to your Arduino. It can also help to get the Ardudroid app from the Google Play store

Example: read a single Analog value

bt = Bluetooth.PickDevicesRfcommName(“”)  
ardu = bt.As (“Ardudroid”)  
FOR i = 1 TO 10  
 val = ardu.Read (1, 1)  
 Screen.ClearLine (4)  
 PRINT “value”, val  
 PAUSE 50  
NEXT i

In the example, we first pick an Ardudroid-compatible device to connect to. Then we read an analog value 10 times.

The available command are

|  |  |
| --- | --- |
| **Command** | **Meaning + Protocol** |
| AnalogWrite (pin, value) | Write analog data to the pin  \*11 {pin} {value} |
| DigitalWrite(pin, value) | Write digital data to the pin  \*10 {pin} {value} |
| Read (pin, value) | Reads data from the Arduino. In the official version of the Ardudroid sketch, this always returns data from Analog pin 0 in the format Analog 0 = {value}  \*13 {pin} {value} |
| ServoAttach (servo, pin) | An addition to the protocol  Attaches servo to pin using Servo.attach (pin)  \*14 {pin} {value} |
| ServoWrite (servo, value) | Writes servo data  \*15 {pin} {value} |
| Write (string) | Writes the exact string to the Ardudroid program. This is useful when you need to extend the Ardudroid protocol. |

The original Arudroid sketch was updated to include Servo commands. An include statement, #include <Servo.h> was added to the top and definitions after #define PIN\_LOW 2

// Additions to support Servo

Servo Servos[10];

#define CMD\_SERVO\_ATTACH 14

#define CMD\_SERVO\_WRITE 15

These additions were added to about line 103 near the end of the void loop() function.

if (ard\_command == CMD\_SERVO\_ATTACH){

Servos[pin\_num].attach(pin\_value);

// the 'pin' is the servo number

}

// 'pin' is the servo number and 'value' is the value.

// often values are 0..180

if (ard\_command == CMD\_SERVO\_WRITE) {

Servos[pin\_num].write (pin\_value);

}

## Infineon DPS310

The Infineon DPS310 Pressure Sensor kit is a small, battery power combination pressure sensor, temperature sensor and altimeter mounted onto the IFX\_NANOHUB Bluetooth-enabled carrier board. [https://www.infineon.com/](https://www.infineon.com/cms/en/product/sensor/capacitive-pressure-sensor-for-consumer-applications/DPS310/productType.html?productType=5546d462525dbac4015312b96a743801)

In the example, a raw Rfcomm (serial port) device is picked. The DPS310 is mounted on an IFX\_NANOHUB and shows up in Bluetooth with that name. Then the dps310 is created as a specialization of the raw Rfcomm device. Note that there’s no good way for Best Calculator to know which device has which specialization for Rfcomm devices. You just have to write the code correctly.

CLS BLUE

PRINT “Altitude from the DPS310”

bt = Bluetooth.PickDevicesRfcommName ↲

(“IFX\_NANOHUB”)

dps310 = bt.As (“DPS310”, ↲  
 “Dps310Altitude”, “Dps310Pressure”, ↲  
 “Dps310Temp”)

FOREVER

FUNCTION Dps310Altitude (dps310, value)

Screen.ClearLine (4)

PRINT “Altitude”, value

END

FUNCTION Dps310Pressure (dps310, value)

Screen.ClearLine (5)

PRINT “Pressure”, value

END

FUNCTION Dps310Temp(dps310, value)

Screen.ClearLine (6)

PRINT “Temp”, value

END

The sensor is supported with a generic Rfcomm Bluetooth interface, but it’s much easier to use the DPS310 specialization. The specialization takes in up to three functions, one each for the altitude, pressure and temperature readings. The altitude is in nominal meters above sea level, the pressure is in mBar and the temperature is in degrees Celsius.

When the specialization is made, three function names are passed in, one each for the altitude, pressure and temperature data. You can also just pass in a blank string (“”) for any function that you don’t need a callback for.

An explanation of the DPS310 raw Bluetooth is at <http://sunriseprogrammer.blogspot.com/2017/07/infineon-sensor-hubfiguring-it-out.html> . An interesting set of contests for the Infineon was run by the <http://Hackster.io> team; the set of samples is at <https://www.hackster.io/contests/Infineon> .

## C:\Users\toomr\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Littlebot.pngSlant Robotics LittleBot

The LittleBot from Slant Robotics is a small, cute, Arduino-based robot kit released in 2017 from a Kickstarter [campaign](https://www.kickstarter.com/projects/slantrobotics/littlebot-a-fun-arduino-robot-for-students-and-beg) and now supported on their [web site](https://www.littlearmrobot.com/uploads/3/8/4/8/38484491/walteros_05.2.ino) at <http://littlearmrobot.com> . The brains of the robot are a small Arduino-based controller that can be controlled with a built-in Bluetooth serial port.

Before you can use the LittleBot you need to download the Walteros ([walteros\_05.2.ino](https://www.littlearmrobot.com/uploads/3/8/4/8/38484491/walteros_05.2.ino)) sketch for the robot from their [downloads](http://www.littlearmrobot.com/downloads.html) page. Install it using an Arduino IDE (Integrated Development Environment). When I programmed mine, I had to remove the Bluetooth board for the USB serial connection to work correctly. I also had to manually set the Arduino type (it’s an “Arduino Nano”)

The Bluetooth connection is “HC-06”

The robot’s Bluetooth command language consists of a series of numeric commands with optional arguments. Each correct command is responded to with a single “d” character (“d” stands for “done”).

The commands are:

|  |  |
| --- | --- |
| **Command** | **Meaning** |
| 1 <left> <right> <grip> | Set the robot wheel speed (the left and right) and the gripper position. Common speeds are 20 and 30 (the code says that 30 is “half speed”) |
| 222 | Stops the motors. The gripper position doesn’t change. |
| 256 | Autonomous mode. The robot will move around at will. |

Unfortunately, there isn’t a command that returns the current ultrasonic detection value.

A very simple program to control the robot. It connects to the robot, switches into autonomous mode for about 20 seconds, and then turns the robot off.

bt = Bluetooth.PickDevicesSppName (“\*”)  
bt.ReceiveString (“BtRecv”)  
bt.Send (“256\n”)

FOR time = 0 TO 10  
 PAUSE 50  
 now =DateTime.GetNow()  
 Screen.ClearLine (2)  
 PRINT “TIME”, now.Time  
NEXT time

FUNCTION BtRecv (device, data)  
 CONSOLE “RECV:”, data  
END

## C:\Users\toomr\AppData\Local\Microsoft\Windows\INetCache\Content.Word\BluetoothSPP.JPGRfcomm (Serial-port) Bluetooth

Some Bluetooth devices (like the Infineon IFX\_NANOHUB and the Slant Robotics devices) work entirely using the Bluetooth Rfcomm mechanism. This is also called the Serial Port Profile (SPP).

The Rfcomm fundamental communication is that you can send data (often a string) to the device at any time, and it can send you information at any time. Often the data looks like it’s individual packets – for example, you might send a command as one string and get a string in return. However, the underlying protocol is not one-for-one.

The little example sends a command to an Rfcomm device and prints the results.

bt = Bluetooth.PickDevicesRfcommName (“\*”)  
bt.ReceiveString (“BtRecv”)  
bt.Send (“$hello\n”)  
FOREVER  
  
FUNCTION BtRecv (device, data)  
 PRINT “RECV:”, data  
END

There are device.As (“name”) specializations for selected Rfcomm devices like the IFX\_NANOHUB and the Slant Robitics devices.

## Using Sensor.Camera and making images

This is an experimental feature. You can expect that the API will change or be removed in future versions.

Best Calculator is able to grab images from the built-in camera (front or back) and display the results directly, display a filtered version, and call a callback function with a color analysis of the image.

### The simplest camera program

The most basic program lets you display the camera. To do this you need to

1. Make an *image* on a graphics screen. You also have to make the graphics screen.
2. Ask for the sensor camera
3. Tell the camera to display onto the image
4. Tell the camera to start

REM Very simple camera program

CLS BLUE

PRINT “Simple camera program”  
  
REM graphic size is x, y, h, w  
g = Screen.Graphics (50, 50, 400, 600)  
REM image size is x1, y1, x2, y2  
img = g.Image (0, 0, 600, 400)

REM Cameras include front, rear, face  
camera = Sensor.Camera(“front”)  
camera.Image = img  
camera.Start()

FOREVER

The resulting camera view uses hardware acceleration to display a high-quality image. This image cannot be analyzed or modified.

### Using Analyze to modify the camera image

You can make a separate image from the preview, modifying the result. This is done by making an analysis object and setting its properties. One of the most important properties is the View property. This is an image (from a graphics.Image(x1, y1, x2, y2) method call) that the analysis results will be displayed on.

The Analysis can map the incoming red, green and blue values from their original values (from 0 to 255) into a new set of values based on a set of points. This is done with a series of analysis.AddPoint (“<channel>”, input, output). The channel is “r”, “g” or “b”

The default points are (0,0) and (255,255). This is the “don’t make any changes” points.

To turn off one of the channels (a channel is one of the red, green, and blue values), make a map that is (0,0) and (255,0). This mapping will turn any input value (0 to 255) to 0.

To make a “posterized” channel, add in a series of points which are “flat”. For example, to make a three-channel posterized channel where each pixel is either 0, 128 or 255, use these set of points: (0,0) (85,0) (86,128) (170,128)(171,255)(255,255)  
Each value from 0 to 85 will map to 0; each point from 86 to 170 will be 128, and each value higher will be 255.

In the simple “red” example, only the red (“R”) values are sent through to the final image. This is done

REM Make an image to display the analyzed  
REM image on  
analysisg = g.Image (0, 200, 300, 400)

REM Make a camera analysis that wil  
REM pass the red values through but  
REM remove the green and blue.

analysis = camera.Analyze ()  
analysis.Image = analysisg  
analysis.AddPoint (“r”, 0,0, 255, 255)  
analysis.AddPoint (“g”, 0,0, 255, 0)  
analysis.AddPoint (“b”, 0,0, 255, 0)

### Example: HTML color from the camera

In this example, we display the rear camera, grab an analysis from the central portion, get the average HTML color (in the standard #rrggbb format) and use that color to change the background of the graphics screen.

CLS BLUE

PRINT “Show HTML Color”

REM Step 1: make the camera and preview

cam = Sensor.Camera (“front”)

grf = Screen.Graphics (50, 50, 340, 440)

view = grf.Image (20, 20, 420, 170)

cam.Start()

cam.Image = view

REM Step 2: make the analysis image

REM and set up callback

a = cam.Analyze()

REM TODO: Radius, CX and CY

a.Radius = 0.25

a.CX = 0.5

a.CY = 0.5

a.AnalysisW = 64

a.AnalysisH = 64

viewanalysis = grf.Image(20, 190, 420, 320)

a.Image= viewanalysis

a.Function = “HtmlColor”

FOREVER

REM Step three: HTML color output

REM and set the background color

FUNCTION HtmlColor (r, g, b)

color = String.Escape (“color”, r, g, b)  
 REM Output is e.g. #112233

Screen.ClearLine (2)

PRINT color

GLOBAL grf

grf.Background = color

END

## Release Notes (3.10, September 2017)

New devices were added: Ardudroid, Infineon DPS310 and the Slant Robotics LittleBot. These are controlled using the new Rfcomm (SPP) Bluetooth support.

Each Array object now includes a Mean field like the SumOfSquares field.

The circle object now takes an optional yradius value. If both the xradius and yradius values are given, the result is an ellipse. The Ellipse method is a synonym for Circle.

The Rectangle object (from g.Rectangle()) now includes a Data member. This is for your use; it will never be examined or changed by BC BASIC. It’s for when you need to save a little bit of information about a graphics object.

The Bluetooth device list methods (Devices, DevicesName and PickDevicesName) all have an Rfcomm version (DevicesRfcommm, DevicesRfcommName, PickDevicesRfcommName). The Rfcomm version will list Bluetooth devices that support the RF COMM (Serial port) protocols. Many Arduino devices, for example, use serial-port Bluetooth chips for communicating with computers.

TODO: Sensor.Camera and the graphics.Image and graphics.Camera

TODO: Update the Sensor.Camera samples

Manual Fixes: section 38.2.5 automatic update. The update also happens during the FOREVER command.

Manual Fixes: the right name for a beLight is “beLight\*” – that’s because the names tend to be, e.g., “beLight 0.2”

The BC BASIC menu item now includes a New Program item. This lets you quickly make a new program (and especially lets you make your first program) without having to trouble with learning the Package system.

String.Escape (“color”, r, b, b) make a string #RRGGBB. The values r, g and b must be 0 to 255. Values less than zero are turned into zero and values greater than 255 are turned into 255.

String.ToUpper(str) and String.ToLower(str) are created

Colors can include strings in HTML hex format like #ADD8E6, the HTML “LightBlue” color.

Added the Circle yradius parameter.

Added PLAY ONNOTE (via PLAY “ONNOTE function”)

Added PLAY WAIT as something that calls callbacks. The MML used in the PLAY command now accepts Unicode sharp ♯ and flat ♭ signs.

Rectangle supports Stroke

Circle can take a 4th arg (and there’s an Ellipse which is a synonym for Circle)