

Microcontrollers and Communication: HC-05 Bluetooth Module

Learning outcomes

1. Gain experience of communication between Arduino and over devices (including other Arduinos) via Bluetooth
2. Use a smartphone as a remote control to trigger the Arduino (turning on and off an LED)
3. Remotely Stream Data from a sensor attached to the Arduino to a smartphone or PC using Bluetooth

Equipment

1 x Arduino Uno & USB cable
1 x breadboard
1 x HC-05 Bluetooth Modules
LED
Resistor
Wires
PC with Arduino Integrated Development Environment (IDE)

Background

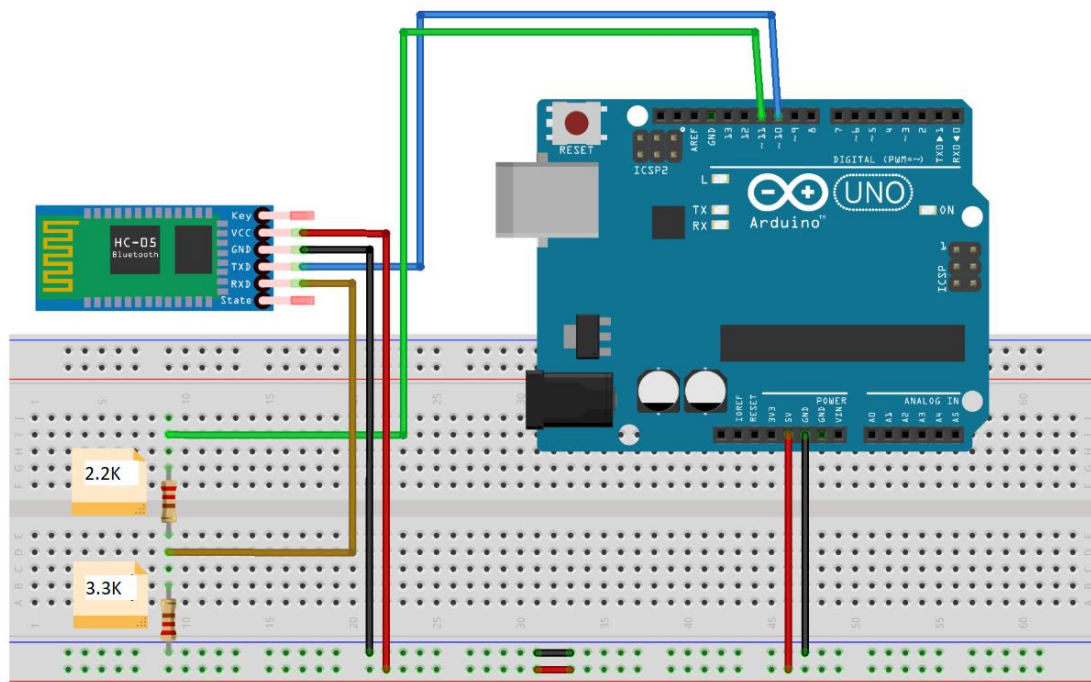
Bluetooth is a useful short-range (30 m) approach to wireless communication. It is a standard feature of smartphones and tablets, and associated technologies (headsets, headphones, speakers etc.) & is increasingly being included in laptop computers. It owes its origins to attempts to find a wireless alternative to serial communication between devices via cables. As such, it presents an opportunity to serve as an alternative to the serial communication between Arduino and computer that usually takes place via the USB cable that connects the Arduino board and PC.



Practical

Initiating serial communication via Bluetooth on the Arduino requires a suitable Bluetooth module (circuit) and a set of dedicated programming commands. For the latter, it is easiest to make use of a **library** containing these commands and other relevant definitions, for example **SoftwareSerial.h**. For the former, we will make use of the **HC-05 module**, illustrated left. The datasheet for the HC-05 is included on the KEATS module page.

The build for the basic Bluetooth-communication set-up is shown on the next page.



You can mount the HC-05 on the breadboard to make connections using male-male cables, or make direct connections using the male-female cables.

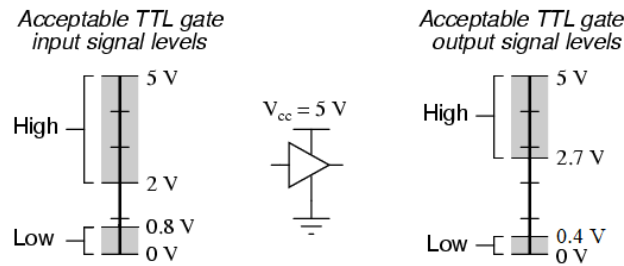
Points to note:

1. Vcc goes to +5V on the Arduino
2. GND goes to GND on the Arduino
3. TXD (or TX) goes to pin 10 of the Arduino
4. RXD (or RX) goes **via a voltage divider** to pin 11 of the Arduino

Why the Voltage Divider?: IC logic types

You can see that the RX connection is not made directly, instead it goes via two resistors, one 2.2kOhms and the other 3.3kOhms. Why is this? The two resistors act as a **voltage divider**, which is to say that a part of the total voltage is dropped across each resistor in proportion to their resistance values. The easiest way to picture this is for the case where the two resistors are the same. In this case, each resistor would drop the same voltage, such that, if we fed 5V into the top of the voltage divider, at its midpoint, the voltage would be $5/2 = 2.5V$. We are using a 2.2k and a 3.3k resistor; thus, it can be seen that we are dropping the voltage at the midpoint by a ratio of 2.2/3.3, or roughly a third. Thus, changing our 5V logic input from the Arduino to around 3.3 V. Why?

This is to do with **logic types**. The first successful implementation of logic gates on IC circuits used the so-called transistor-transistor-logic (TTL) approach, which is to say using transistors to both realise the logic and amplify the logic signal. The particular transistors used are called “bipolar junction transistors”. The logic output was set at +5V for 1/HIGH and 0V for 0/LOW. The illustration below shows the range of voltages accepted as HIGH or LOW by TTL logic circuits.

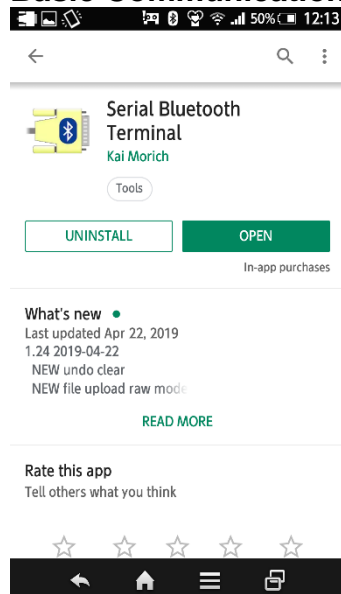


The problems with TTL logic are high power consumption and, for some applications, relatively-slow switch speed (HIGH to LOW or vice versa). These problems were addressed with a later logic architecture called CMOS that makes use of a different type of transistor – the “field effect transistor”. CMOS-based logic consumes less power, has faster switching times, and is generally more stable than TTL. The principal practical difference between the two is that **CMOS uses 3.3V as a logic-level HIGH, while TTL uses 5V as logic-level HIGH**.

The HC-05 module uses CMOS logic. However, the Arduino uses TTL. This is not an issue on the HC-05 to Arduino communication direction, as the Arduino TTL will read 3.3V as HIGH (see illustration above). But it can be a problem in the opposite direction, as the 5V from the Arduino is too high for the CMOS receive pin (actually, in the short term, the pin will happily take 5V, but there is a danger of breakdown, and it is good practice to protect against this even in the short term). Thus the voltage divider.

Research the equation for determining the mid-point output of a two-resistor voltage divider and confirm that the voltage divider you are using transforms the +5V from the Arduino pin 10 down to +3.3V

Basic Communication: the terminal



Before going on to look at the Arduino sketches that will allow us to communicate with the board via Bluetooth, we need to set up the other end of the communication chain. In the basic experiment, you will make use of your **smartphone**. There are many apps that allow for direct communication with Bluetooth-enabled devices. These are called “terminals” and are analogous to the **command prompt** on the PC. **Download and install a suitable Bluetooth terminal onto your smartphone.** There are many to choose from. These notes are written with the **Serial Bluetooth Terminal** app, as illustrated left, in mind; however, they will also apply to other equivalent apps.

Arduino Sketches

Power up the board and Bluetooth circuit. The Bluetooth module light should start flashing. Start up the Serial Monitor. In the first instance, having built the circuit above, copy over the **TPE_SimpleBlue** sketch into the Arduino IDE and upload this to the board.

```
// taken from: https://exploreembedded.com/wiki/Setting_up_Bluetooth_HC-05_with_Arduino
//

#include <SoftwareSerial.h>
SoftwareSerial EEBlue(10, 11); // RX | TX pins (TX pin 10, RX pin 11 via voltage divider 2.2k-3.3k)

void setup()
{
    Serial.begin(9600);
    EEBlue.begin(9600); //Default Baud for comm, it may be different for your Module.
    Serial.println("The bluetooth gates are open.\n Connect to HC-05 from any other bluetooth device
with 1234 as pairing key!.");
}

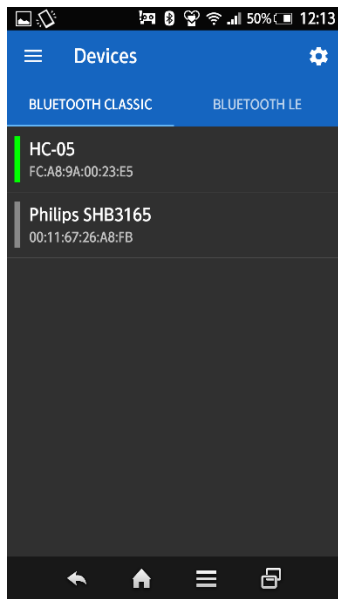
void loop()
{
    // Feed any data from bluetooth to Terminal.
    if (EEBlue.available())
        Serial.write(EEBlue.read());

    // Feed all data from terminal to bluetooth
    if (Serial.available())
        EEBlue.write(Serial.read());
}
```

As can be seen, we are:

1. Including the SoftwareSerial library (#include <SoftwareSerial.h>) **note the absence of a semi-colon at the end of the command**
2. defining an instance of software serial control called "EEBLUE" and telling the board what pins it uses(SoftwareSerial EEBlue(10, 11);)
3. as well as setting up the serial monitor at 9600 baud, we are also setting up the Bluetooth serial communication at the same rate EEBLUE.begin(9600);
4. Making use of the same .write and .read commands for the serial monitor and the Bluetooth serial communication
5. If everything goes as it should you will see on the serial monitor the message:
"The bluetooth gates are open. Connect to HC-05 from any other bluetooth device with 1234 as pairing key!."

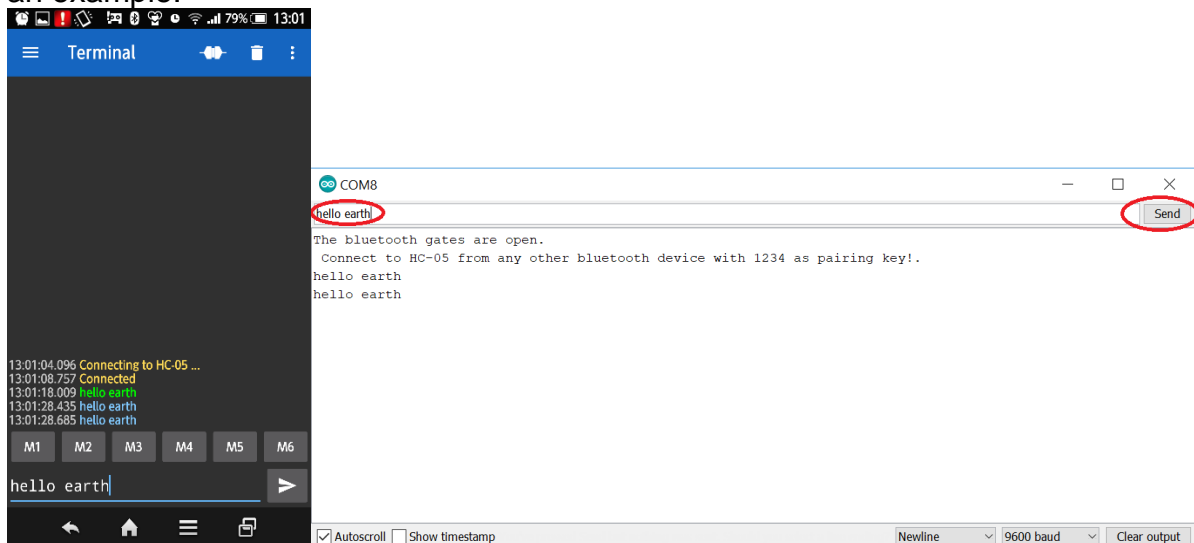
Once you have uploaded the sketch, it is time to pair the phone and HC-05. Turn on your phone Bluetooth and look for devices to pair with. **If every group has powered-up their Arduinos at the same time, you will see a plethora of HC-05s to pair with. A little organisation with respect to the sequence or timing of powering-up the boards will help matters. The pairing key is 1234.**



Having paired your smartphone and the HC-05, you can now open the Bluetooth terminal on your phone and select the paired HC-05. In Serial Bluetooth Terminal app this is done by selecting “Devices” and then the HC-05 module, as left.

Returning now to the “terminal” screen, you should see a message that you are now connected (see illustration below).

You should be able to enter a word or words into the command prompt and send this, like a SMS message, to the Arduino. This will be displayed on the Arduino Serial Monitor. Similarly, you can enter a word or word into the command prompt of the serial monitor and then hit “send”, and this should appear on the screen of the Bluetooth serial monitor on your smartphone, as illustrated below for “hello earth” as an example.



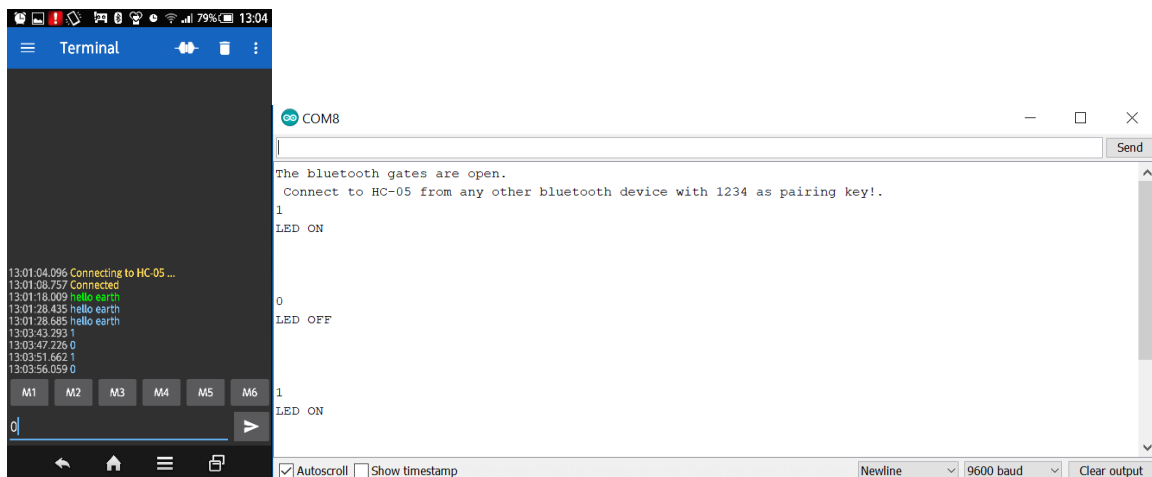
Using the Bluetooth terminal to control a device

Add to your circuit an LED and resistor connected, via the resistor, to pin 13 of the Arduino, as you did in the initial Arduino experiment.

We will now use the Bluetooth Serial monitor on your smartphone to switch on and off this LED. To do this, copy over the **Bluetooth_LED** sketch and upload this to the Arduino.

Read through the code and, by comparing with TPE_SimpleBlue, find the additional commands that read from the Bluetooth and execute the commands that it sends.

If you have been successful, entering “1” into the terminal will turn on the LED, “0” will turn off the LED, and the serial monitor will show what you have entered.



Remotely Streaming Data from a sensor attached to an Arduino to a PC.

Of more direct interest to us is the ability of adding Bluetooth to allow us to remotely stream data from an Arduino build. This is illustrated for the case of the TEMT6000 light sensor below.

Keeping all the previous connections to the HC-05 in place, reattach the TEMT6000 light sensor to the Arduino (Vcc to 5V, GND to ground, OUT to analog pin A0). There is a sketch for streaming data from the TEMT6000 via Bluetooth on the KEATS module page. It is called **TPE_BlueStream** and is reproduced below.

```
// taken from: https://exploreembedded.com/wiki/Setting_up_Bluetooth_HC-05_with_Arduino
// modified J Barras 0719

#include <SoftwareSerial.h>
SoftwareSerial EEBlue(10, 11); // RX | TX pins (TX pin 10, RX pin 11 via voltage divider 2.2k-3.3k)

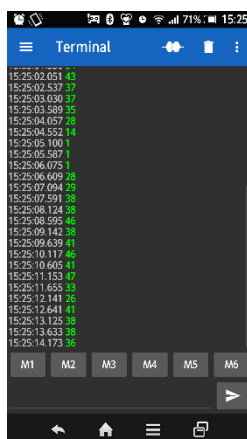
void setup()
{
    Serial.begin(9600);
    EEBlue.begin(9600); //Default Baud for comm, it may be different for your Module.
    pinMode(A0, INPUT); //for reading the sensor
    Serial.println("The bluetooth gates are open.\n Connect to HC-05 from any other bluetooth device
with 1234 as pairing key!.");
}

void loop()
{
    int val=analogRead(A0);
    Serial.println(val);
    EEBlue.println(val);
    delay(500);
}
```

It can be seen that we have set-up the HC-05 as before. The new elements are:

1. Setting up pin A0 and reading from it to an integer we have called "val".
2. A print command associated with the instance of the Bluetooth module: `EEBlue.println(val);` this is equivalent to printing on the serial monitor, but instead transmitting the data via the Bluetooth

Upload the TPE_StreamBlue sketch to your Arduino. **Then disconnect your Arduino from the PC and switch to powering it from a USB powerbank.**



If you set up the Serial Bluetooth Monitor app on your smartphone again, you should see the readings of the light sensor appear on its display as on the left (cover the sensor with you hand and confirm you can see the readings drop in intensity).

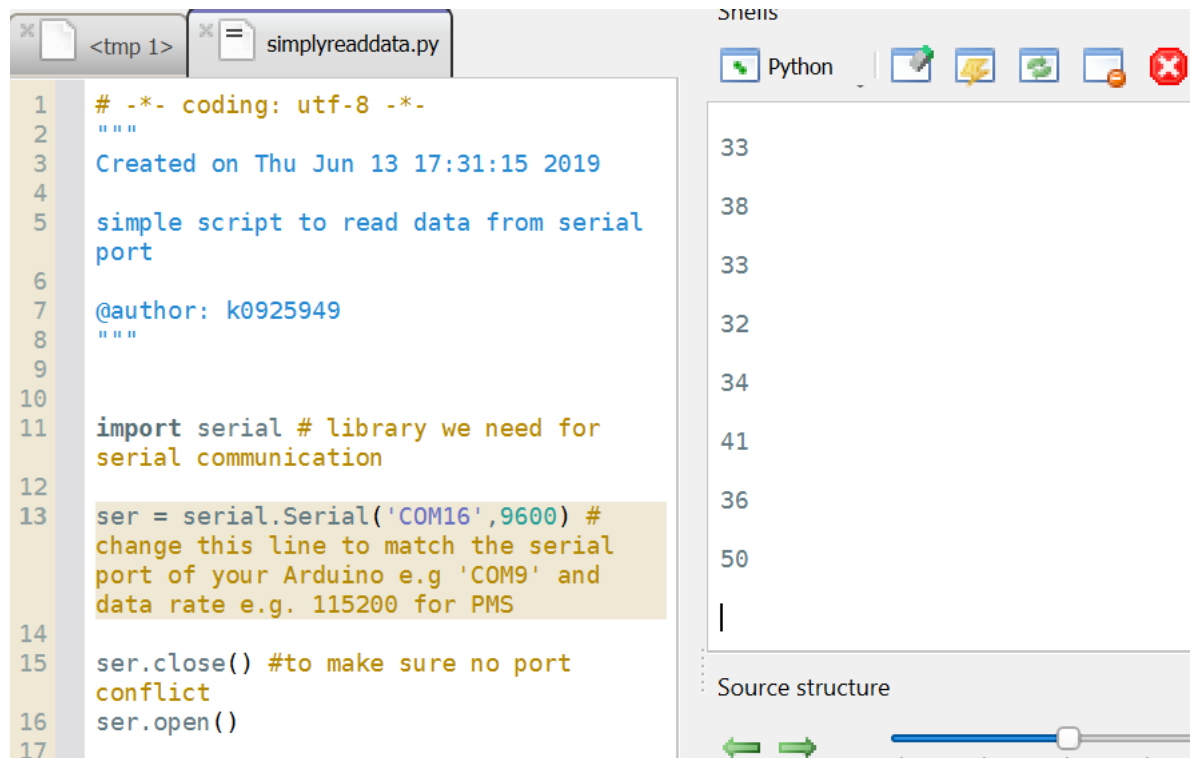
This confirms that we can stream data from the sensor across the Bluetooth remote serial connection.

The next, and final step, is to use PYTHON to do an equivalent operation with the PC.

Reading data from the sensor into Python via the Bluetooth is exactly the same as reading data into Python via the USB connection: the python sketch simply needs to know which port is being using by the Bluetooth.

Either using a laptop with Bluetooth, or a PC with a Bluetooth “dongle”, add the HC-05 to the list of paired Bluetooth devices (remember the pairing code is 1234). Now we need to know which port the Bluetooth is using. The simplest way to do this is from inside the Arduino IDE. If you bring up the Tools > Port list you will see TWO new ports listed (e.g. COM16 and COM17). These are the separate transmit and receive Bluetooth Ports. For streaming data we will make use of only one of these; however make a note of both ports for now.

We can use the simplyreaddata.py script to bring this sensor data into Python, simply changing the port being read to ONE of the two ports used by the Bluetooth (you may have to experiment to find out which one delivers results). If you are successful, you will see in the kernel the data from the sensor start to stream, as below (using COM16 as the Bluetooth serial port).



The screenshot shows the Arduino IDE interface. On the left, a file named 'simplyreaddata.py' is open, displaying a Python script. The script includes a header with encoding and author information, followed by an import statement for the 'serial' library. The main logic sets up a serial connection to 'COM16' at a baud rate of 9600, opens the port, and then closes it. On the right, the 'Serials' monitor window is active, showing a stream of data received from the sensor: 33, 38, 33, 32, 34, 41, 36, 50. Below the monitor, a 'Source structure' section shows a progress bar.

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Thu Jun 13 17:31:15 2019
4
5  simple script to read data from serial
6  port
7
8  @author: k0925949
9  """
10
11 import serial # library we need for
12 serial communication
13
14 ser = serial.Serial('COM16',9600) #
15 change this line to match the serial
16 port of your Arduino e.g 'COM9' and
17 data rate e.g. 115200 for PMS
18
19 ser.close() #to make sure no port
20 conflict
21 ser.open()

```

Serials

Python

33
38
33
32
34
41
36
50

Source structure

This data is now available to be acted upon exactly if received via a physical cable connection.