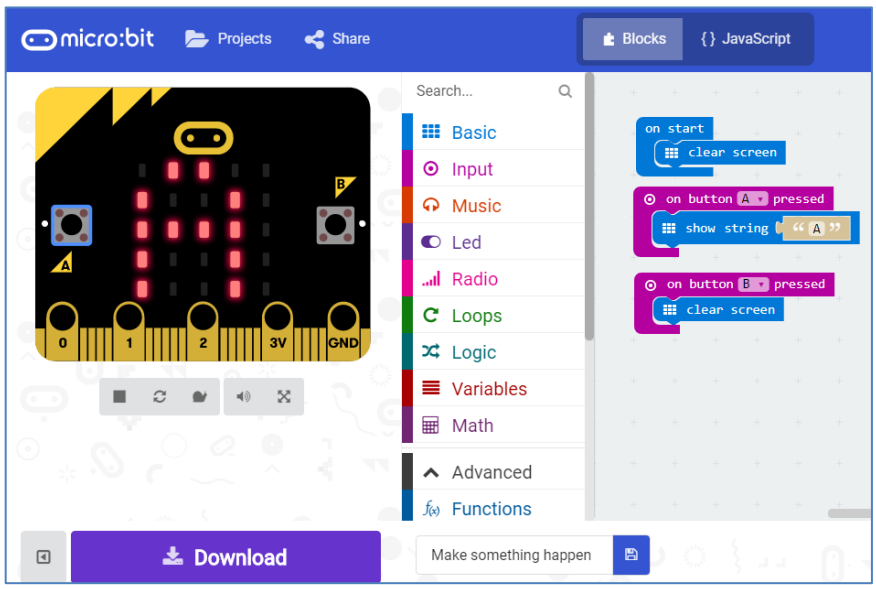
First steps in getting micro:bits and PCs to talk to each other Adrian Oldknow

0. Overview While the BBC micro:bit has been designed to be used as a stand-alone device, it can also communicate with other software, such as *Tera Term*, *Excel* and *Scratch*, exchanging data along a USB cable or wirelessly. This article explores simple ways to do this using the MS *MakeCode* editor in *Blocks* mode. As an example, it demonstrates how to send live sensor data from one micro:bit wirelessly to another, which passes it via USB to the MS *Excel* spreadsheet for graphing and analysis, thus implementing telemetry. Alternative approaches are explored.

1. **Introduction** Along with my other `*First steps*' articles, this piece is written for people who have access to a BBC micro:bit and would like to explore ways to get the most out of it with the minimum of technical hassle. I learned to program in a computer language called *BASIC*, which was specifically written for General Studies' students to pick up quickly and get things happening straight away. Along with many people of my generation, I also became a fan of the *Logo* graphical programming language, developed by Seymour Papert, to widen computer access to children and beginners. In his seminal book, *Mindstorms*, Papert explains his vision of a community of learners, young and old, sharing their discoveries and inventions like Samba dancers do. So that is the spirit in which this article is produced.

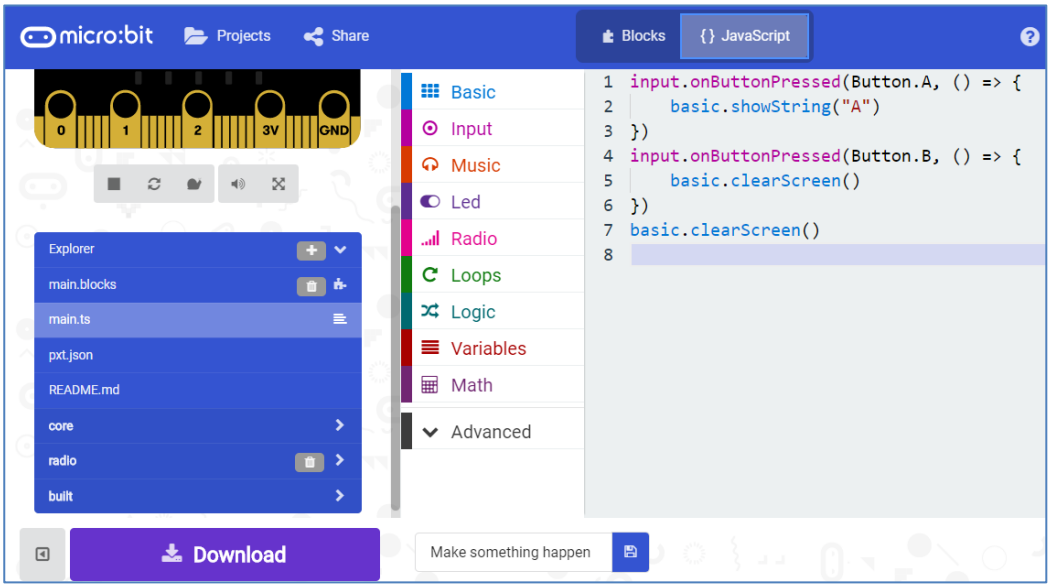
2. **Starting from Scratch** Computing is now in the English schools' National Curriculum and Primary School students learn to code in *Scratch*. Nowadays people tend to use the word *coding* in preference to *programming*, when really coding is only a part of programming. Codes were developed to transmit messages swiftly and securely. It is the substance of the message which matters most. An often used example of coding is getting a computer to display text by a command like: *PRINT "Hello World"*. Programming languages, such as *BASIC*, *Logo* and *Scratch*, enable us to invent our own procedures, technically called *algorithms*, to get computers to solve problems or perform complex tasks. It is easy to learn the words, but the important thing is to develop the skills to use them to say something meaningful. End of sermon! *Scratch* takes the words from *Logo* and puts them in an environment, known as *Build Your Own Blocks* (BYOB), where it is easy to drag them to develop structures. That is the basis of the way that non-experts can develop programs for the BBC micro:bit using editors such as the current Microsoft *MakeCode*.

Here is a very simple example. My program is called *Make something happen*. It has three very small blocks of code. The first is what to do when you plug in the batteries or press the *Reset* button. The second is what to do when you press the **A** button and the third is what to when you press the **B** button.

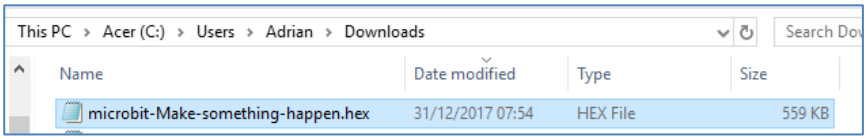


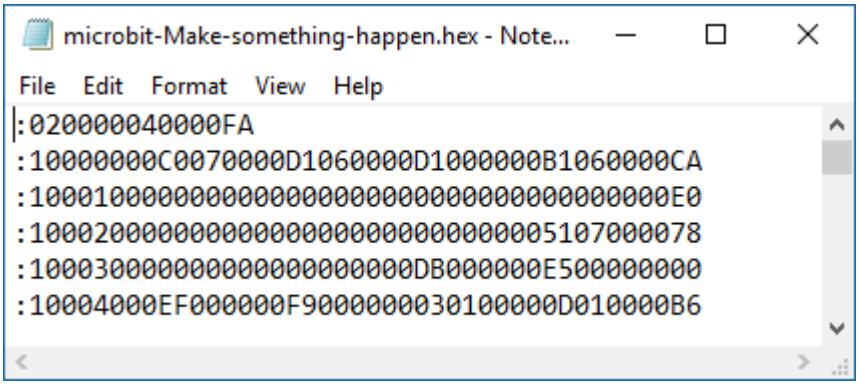
The blue items in the program come from the *Basic* menu, and the purple ones from the *Input* menu. You can build and test your programs in the editor using the simulated micro:bit on the left. So, if you have a computer connected to the internet, you can create programs for a free, virtual micro:bit! There are also editors for mobile devices, like phones and tablets running Android and Apple operating systems.

3. **Digging deeper** We need to understand the key difference between the design of computers, such as my *Windows* 10 laptop, and the BBC micro:bit. The micro:bit is a device known as a *microcontroller*, which is designed to be built into systems to make them work on their own. That is what turns a fridge into a *smart* fridge. It can respond to changes in its environment. Our little program isn't like the programs I am used to writing for my computer. It isn't a sequence of instructions. It is a set of rules on what to do in different circumstances. In this case, how to respond to different *events* – such as the pressing of a button. It is also designed to be able to operate on its own without a computer. So, once the program has been transferred to the micro:bit, it can be disconnected and will work as an *autonomous* device. The micro:bit is a simple relative of the incredibly smart engine management system which looks after my Jaguar car's V8 engine. This responds to all kinds of events, monitoring its environment with a large number of sensors. You may already have played with some of the sensors built into the micro:bit, such as light intensity and acceleration. Now let's return to our simple program. So far we have worked in *Blocks* mode. Try switching to *{} JavaScript* mode. This gives us the same set of instructions but in a linear, text, format. **Same ideas, different code**. One of the advantages of this format is that it makes it easier to store and print out. But it also uses a lot of unfamiliar symbols which make it hard to read! In the end, though, the micro:bit can't understand either the *Blocks* or *JavaScript* versions of our program. It needs to receive a version coded into a series of numbers which its, so-called, *machine language* can understand. The machine in question is called an *ARM Mbed processor*. At the heart of every computing device is something called a *processor,* which sends and receives electronic signals as pulses of high and low voltages. These are represented by the binary digits 0 (low) and 1 (high). A binary digit is called a *bit*, as in micro:*bit*. Computing devices are designed to be able to cope with several bits at time, called *bytes*. A typical byte, such as 10010101, has 8 bits.



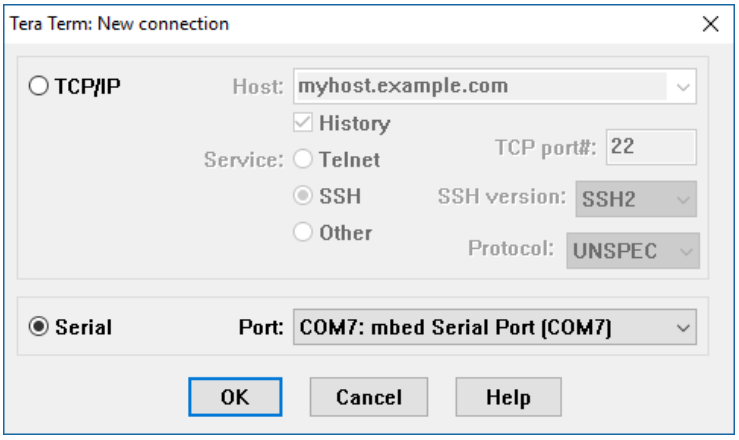
The smallest 8-bit number is 00000000 or 0 and the largest is 11111111 or 255. We are used to working in a *decimal* system where we use powers of 10, such as hundreds, thousands, millions etc. The *Mbed* uses the *hexadecimal* (or *hex*) system to send numbers, i.e. powers of 16. So we need 16 different symbols for the numbers 0 to 15. 0 to 9 will do fine. Then we can use A, B, C, D, E and F for the numbers ten to fifteen. So let's take a number like 11010111. This splits into two *nibbles*, 1101 and 0111. In decimal, 1101 = 1x8 + 1x4 + 0x2 + 1x1 = 13 and 0111 = 0x8 + 1x4 + 1x2 + 1x1 = 7. In hex, 13 = D and 7 = 7. So the hex code for binary 11010111 is D7. When you press the *Download* button in *MakeCode*, your program is converted (or *compiled*) into a hex file which gets stored in your Downloads folder. This is recognised by *Windows* as a text file which can be opened in MS *Notepad*. This is a long file, and just the first few lines are shown here. Each pair of hex symbols, like FA, CA, E0, 78, 00 and B6, corresponds to something meaningful in the *Mbed* machine code.

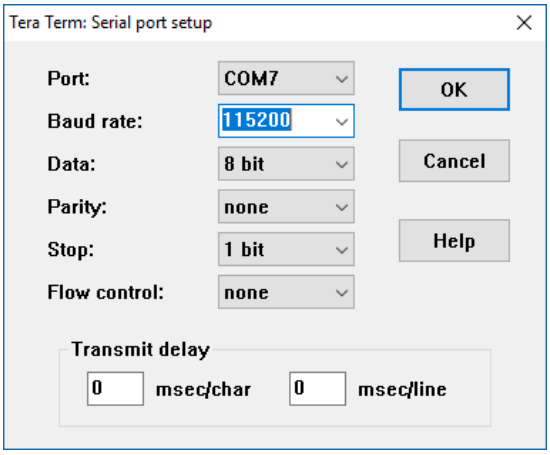




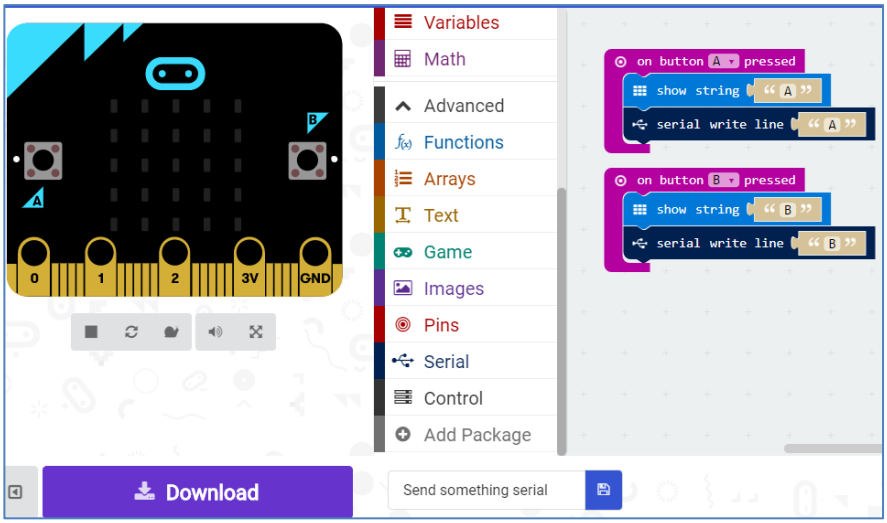
4. **Transferring information to and from a micro:bit** If, like me, you use the *MakeCode* editor on a Windows laptop, you will use a USB cable to send the hex file from the PC to the micro:bit (m:b). The large end plugs into a USB port on the computer, and the small end into the Micro USB port on the m:b. As soon as you make the connection, *Windows* recognises the m:b as an external memory device, such as D:, just as if you had plugged in a memory stick or SD card. USB stands for *Universal Serial Bus*, which transfers data as a series of 0s and 1s along a single wire. The alternative method was called *parallel*, and used a much larger cable to send each of the 8 bits of every byte of information down a separate wire. This was mainly used for printers, and older computers would normally have a *parallel port* built in. If you write your programs on a mobile device, like an Apple tablet or Android phone, you don't use a physical connection along a wire to transmit data with a m:b, you use a wireless *Bluetooth* connection instead. I would like to be able to take data sensed from a micro:bit and send it wirelessly to my laptop, but I'll do this in stages. First we'll look at sending serial data to a PC. Then how Microsoft has developed its free *Project Cordoba* plug in for *Excel* to enable two-way exchange of data with m:b using serial connection. The we'll explore sending data wirelessly from one m:b to another. Then we'll put the pieces together and send data from m:b A wirelessly to m:b B, which then passes it over to *Excel* via USB. Finally we'll see how Clive Seager has cracked exchanging data in both directions with m:b and PC running *Scratch*.

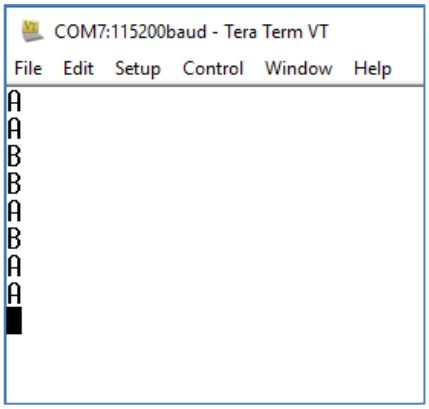
5. **Reading serial data from a micro:bit on a PC** This uses a free piece of software called a *terminal*. Instructions on how to set up the serial connection are on the Micro:bit Serial Library pages. The serial driver is available from the ARM mBed pages. My drivers use serial port COM7 with a baud rate of 115200. I also followed the instructions to install the *Tera Term* software to provide a text window to view the incoming data. After installing and starting *Tera Term* you need to enter the correct values for the Serial port and Baud rate.



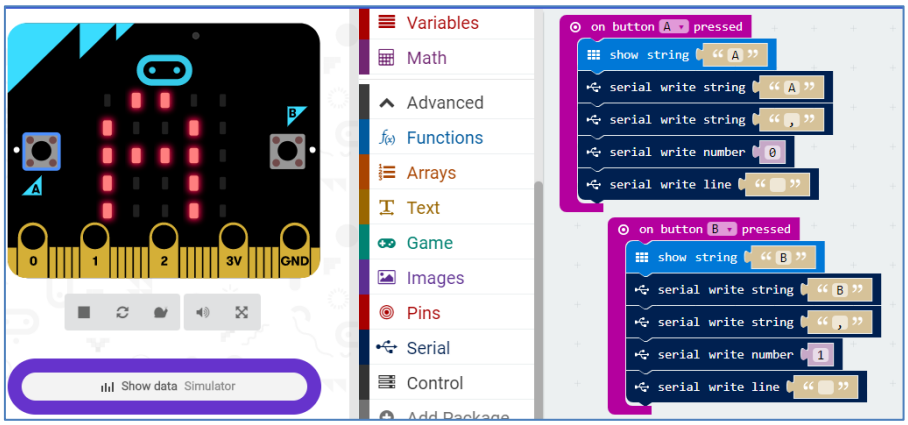


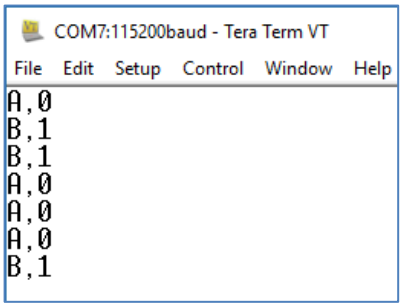
We now need to adapt our little program to include some serial commands. I have played around with the *Tera Term* Setup options to reverse the colours for both text and background, and to make the Font bigger.





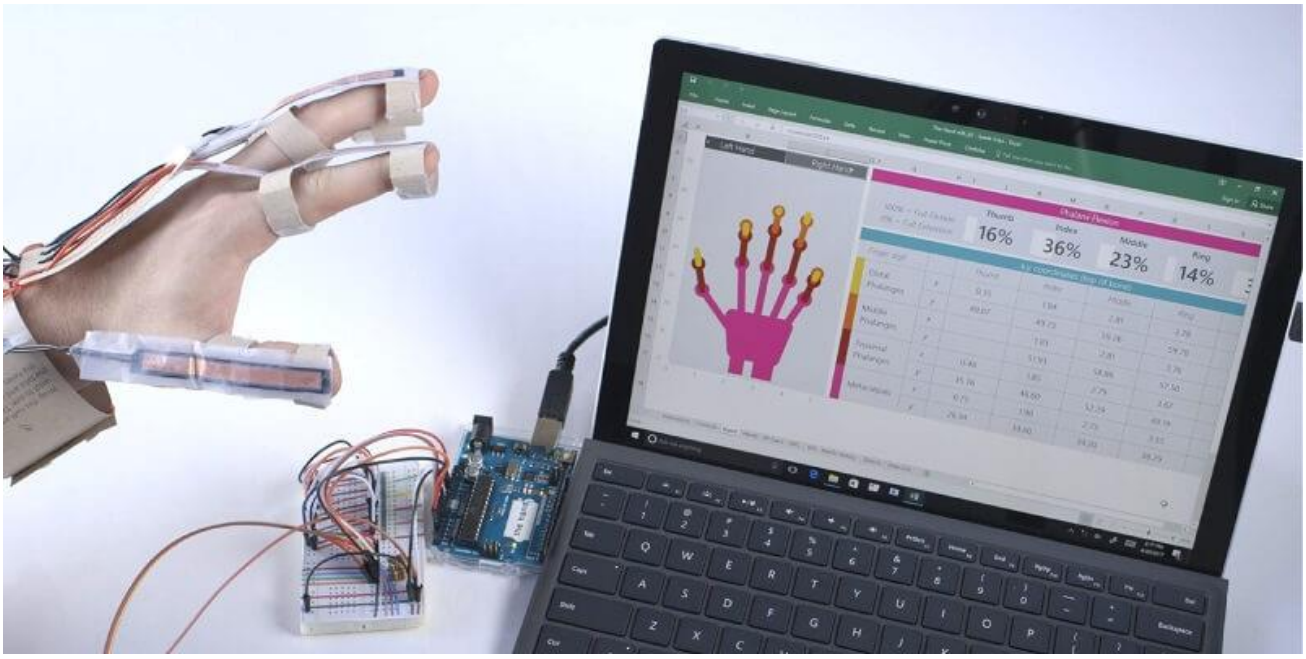
Now we'll try a slight variation to see how to send a more complex *packet* of information. Now pressing **A** builds up a line consisting of the text string "A", a comma "," to separate it from the next piece of data, and a numeric value "1". Similarly for **B**.



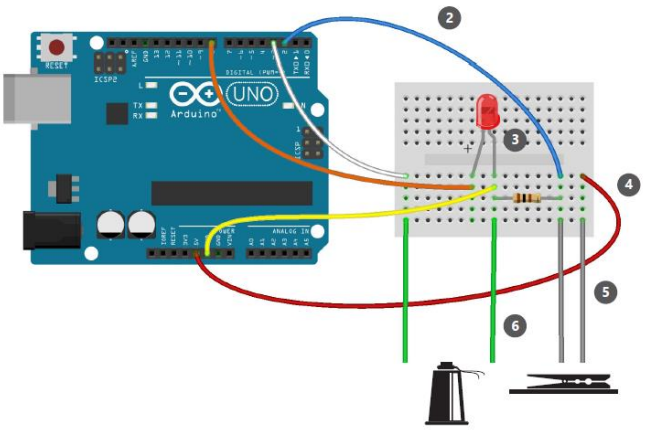


Here is the output as received by *Tera Term*. The data can then be copied and pasted into other applications. The next step is to explore how we can send the data directly into an *Excel* spreadsheet.

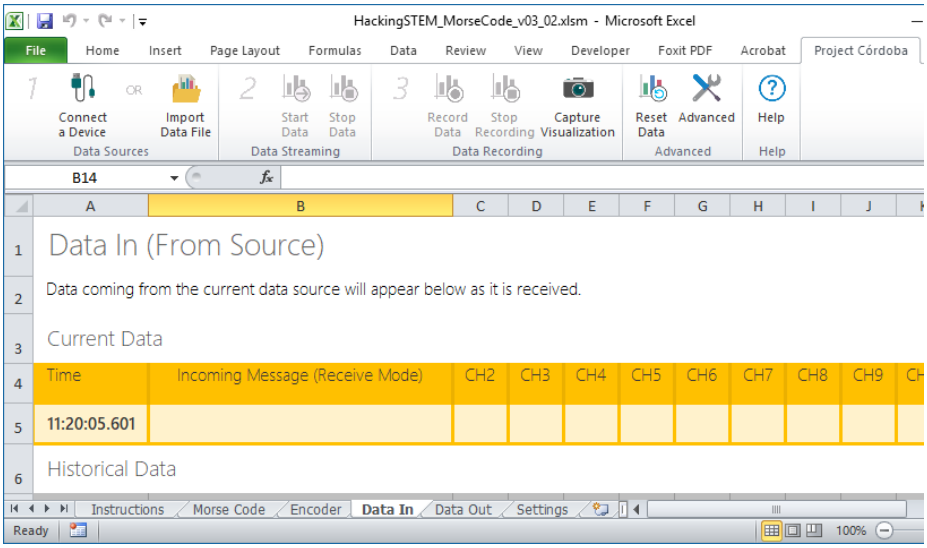
6. **Hacked *Excel* – *Project Cordoba*** At BETT 2017, Microsoft had a large stand in the STEAM Village, next to the Micro:bit Education Foundation stall. A large team were demonstrating a range of projects designed for the US Middle School curriculum called *Hacking STEM*. These were built around circuits including *Arduino Uno* micro-controllers, connected by USB cables to Windows PCs running *Excel*. The image shows a glove made from cardboard and conductive threads being used to sense hand movements. The data is relayed from the *Arduino* to *Excel* which uses it to control a live graphical simulation. This robotic hand project is one application of a free plug-in for *Excel* called *Project Cordoba*, developed as part of Microsoft's `Garage project'. I tried to find a simpler project to test the system myself.



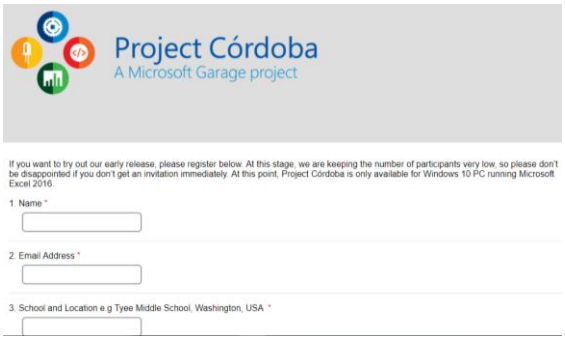
The one I found in the *Hacking STEM* Activity Library was the *Telegraph* project: *Harnessing Electricity to Communicate*. The link took me to the *One Drive* library from where you can download the very clear pdf for the project. Skipping to page 15 I found a simple circuit diagram for a Morse code tapper. Instead of making my own loud-speaker (shown by label 6), I just put a buzzer on the board. Instead of making a clothespin tapper (shown at label 5) I just pushed the end of the red wire (shown at label 4) in and out of the bread-board to make and break the circuit. Fortunately the Arduino code



was available in a convenient file to download from the link on page 14. I followed the instructions and flashed this from the *Arduino IDE* to the *Uno*. Finally I downloaded the *Excel* workbook from the link on page 17. Because I had already been registered with *Project Cordoba* this opened up with the plug-ins tab at the end of the tool bar. So I was able to use it to connect my Arduino and to send and receive data. But when I took a closer look at the Arduino code I was shocked to see how complex it was. I couldn't use it to find a simple way to reduce it to just transmit or receive a single number or string. So I decided to work out from scratch how to get data sent and received using a micro:bit instead of the Arduino. Fortunately this turned out to be much simpler than I expected! The clue was to investigate the final three sheets of the workbook shown on the tabs labelled *Data In*, *Data Out* and *Settings*.



Clicking on the *Help* tab opens up the very limited *Cordoba* manual which runs through some of the set up procedures.
To get a copy of the *Project Cordoba* plug-in for yourself you need to apply using this link.



If successful you will receive an e-mail response in a day or two with instructions on how to download and install *Cordoba,* and the manual.

Happy New year! Welcome to Project Córdoba, the Excel add-in that enables real time data streaming and visualization from physical computing devices! We are excited that you are joining our community and hope you enjoy checking out our STEM units: Harnessing Electricity to Communicate; Analyzing Wind Speed; Increasing Power Through Design; Seismographs; Earthquake Resistant Structures; Building Machines that Emulate Humans and Measuring Speed to Understand Forces in Motion! Here is the link to install Córdoba on your Windows 10 computer: Installation Link. Please remember that you need to have a desktop version of Microsoft Excel pre-installed on your machine to be able to use Córdoba. If you don't have it already, here is a handy link that will take you to where you can download it. We want to make sure that the Córdoba installation process is as easy as possible, so please follow these steps to install it.

Three steps to installing Córdoba

1. Click on this link: Installation Link
2. This link will trigger a download.
3. Please save the file, double click the downloaded file (.exe) and complete the installation process.

How to see if the installation was successful

1. Open a blank Excel workbook.
2. The installation will have created a new tab on your ribbon called “Project Córdoba” in your Excel spreadsheet.

How to use Córdoba

Please find the Project Córdoba user guide attached to this email. The user guide is also available by selecting the ‘Help’ button on the Project Córdoba tab in Excel. Each of the customized Excel spreadsheets to visualize real time data can be found at www.aka.ms/hackingstem.

How to uninstall Córdoba

1. Go to “Control Panel” → “Programs” → “Programs and Features”.
2. Scroll down to “Project Córdoba”.
3. Right click, and select “Uninstall”.

Please remember that you are one of the first few people in the world to get access to this prototype. Your feedback is essential to making Córdoba better. We are eagerly looking forward to working with you to improve its functionality and your experience. Our goal is to evolve Córdoba into a seamless tool that supports you in your classroom. Please share any feedback or suggestions, as we want to hear it all.

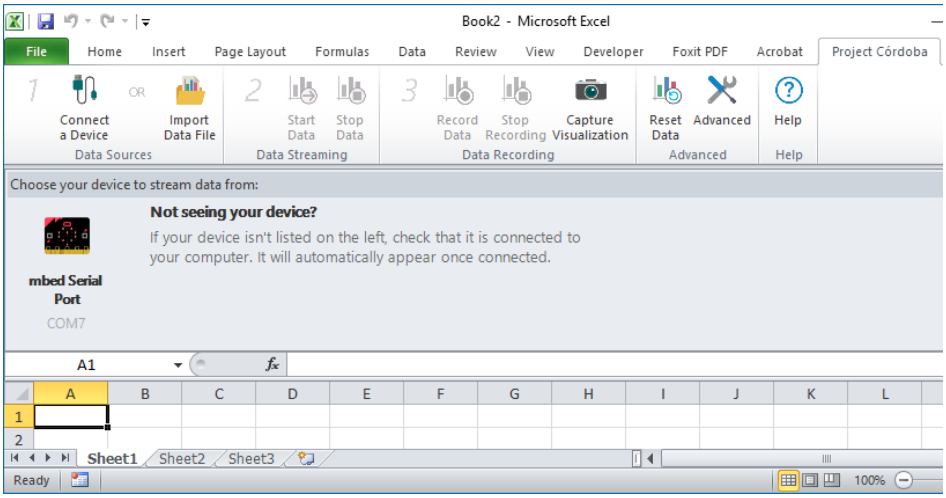
- Email us at hackSTEM@microsoftcom or
- Share your comments on the Project Cordoba Forum

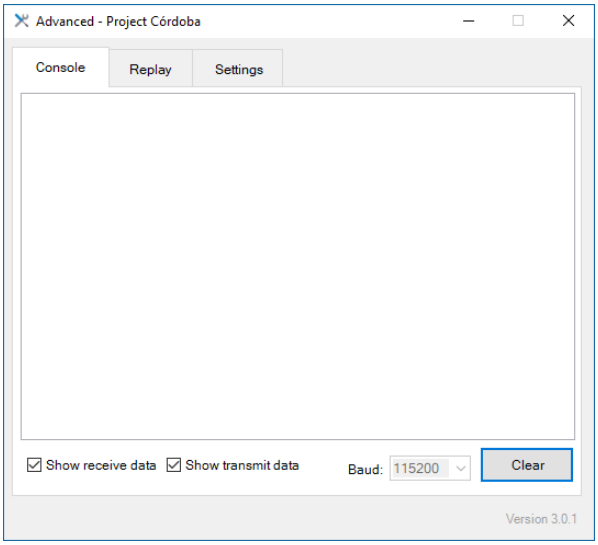
We look forward to learning from you! Kind regards, Hacking STEM team @hacking_STEM

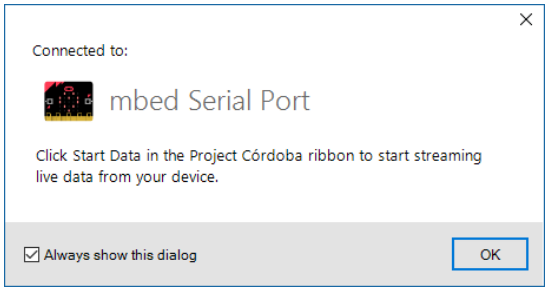
Microsoft

Terms of Use

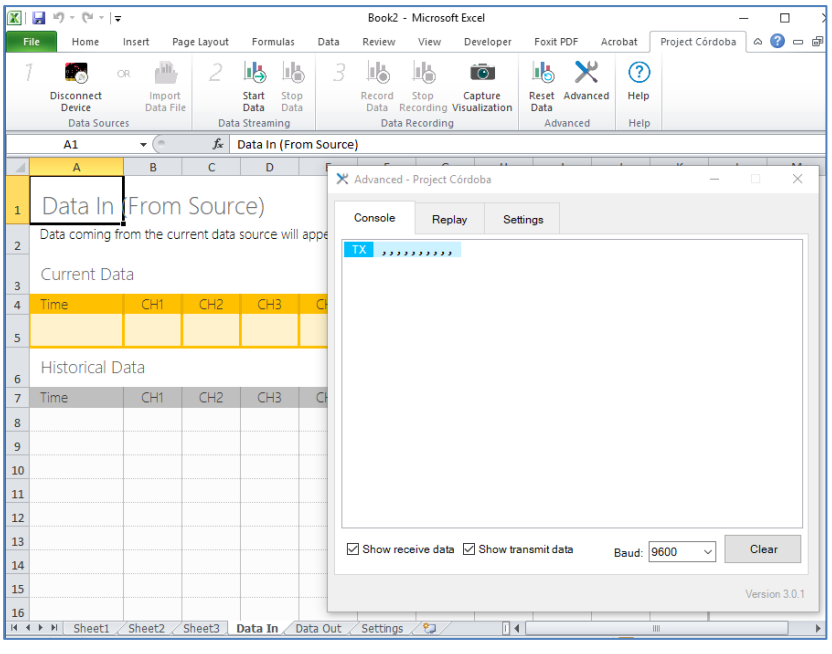
7. **Hacking *Cordoba* and *Excel* to receive serial data from a micro:bit** The first step is to open a **New** *Excel* workbook from the *File* menu on the Morse Code *Excel* workbook. I was a bit surprised to find that the vital extra 3 worksheet tabs (*Data In*, *Data Out* and *Settings*) were now missing, but I ploughed on regardless to see if I could set up the sheet to talk to my micro:bit. After plugging in the m:b, I used the *Connect a Device* command from the *Data Sources* menu. I was pleasantly surprised to find the micro:bit was recognised as an *mbed Serial* device, and that it told me that it was connected to the COM 7 port. It's not obvious what to do next, but the clue is try the *Advanced* tab. This opens up the *Cordoba*'s Console which acts like the *Tera Term* terminal to show you the data stream. It also allows you to enter the correct baud rate, which is 115200 bits per second. Now click on the *mbed Serial Port* icon and you should get a message.



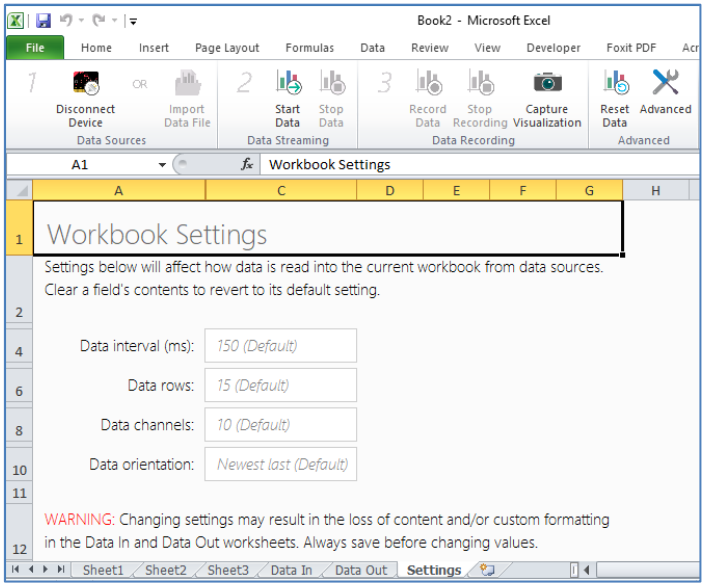




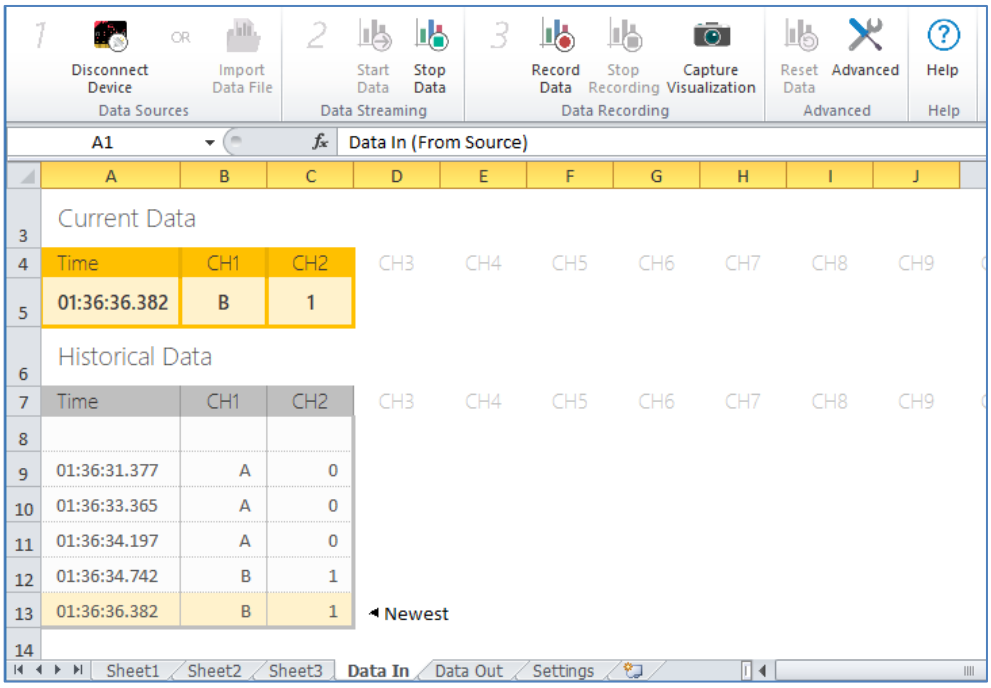
If you click on *OK* you can first use *Start Data* and then *Stop Data* from the *Data Streaming* tab. Now you find that three extra worksheet tabs appear for *Data In*, *Data Out* and *Settings*. The console shows that *Excel* has transmitted a sequence of 10 commas. Now it's time to take a look at the *Settings* tab to see what control we have over the way data is transmitted and displayed.



The main controls we will use are the number of *Data channels* and the number of *Data rows*. For my little experiment I will only use 2 channels at most. In our *Tera Term* example we sent data in parcels containing a string and a number, so 2 channels will be fine. The number of rows affects how much historical data *Excel* holds on to. I will make this fairly small to start with so we can see how the data-stack builds up when we start receiving m:b data. Let's try 6 for starters!

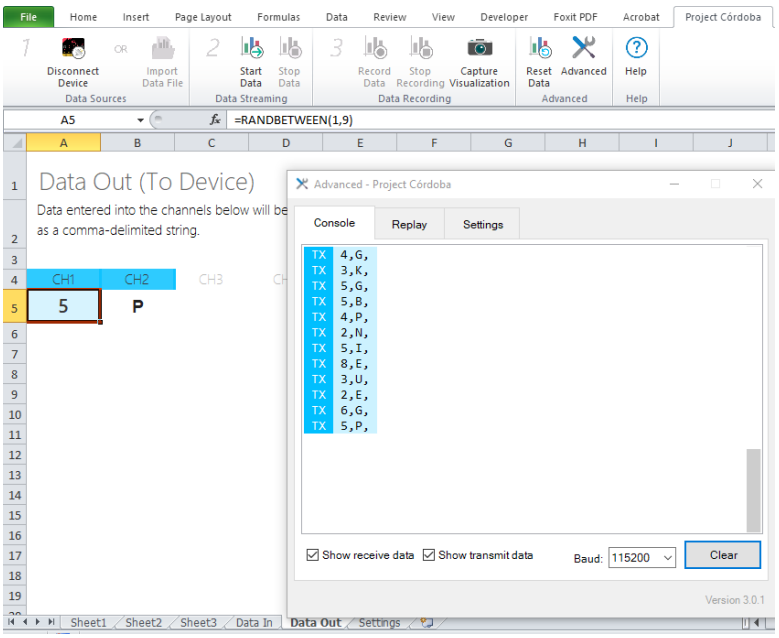


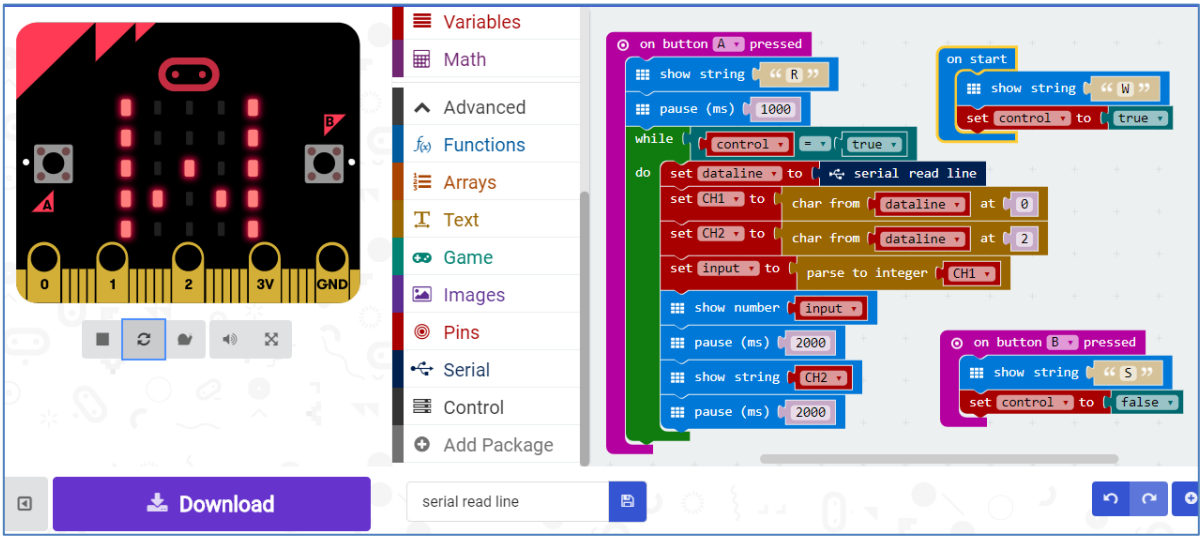
Here is the *Data In* sheet set for use with our latest version of the *Send Something Serial* micro:bit program. The comma delimiter in the packet of data we send from the m:b ensures that the first item gets written to channel CH1 and the second to channel CH2. When you select *Start Data* nothing will happen until you begin pressing the **A** or **B** buttons on the micro:bit. If nothing happens then it may be that you need to use *Advanced* again to make sure the correct baud rate is being used. You should see that the data in the table fills up from row 13. Once all the rows are full the data should scroll so that the latest 6 items are visible.



8. **Sending data from *Cordoba* to micro:bit** First we will set up a new blank sheet for the experiment and get it prepared for micro:bit communication as above. Now we will make it generate a random digit and character to send to the micro:bit. The *Data Out* sheet looks like this:

We just have two output channels as before. In cell A5 we enter the formula: =RANDBETWEEN(1,9) and in B5: = CHAR(RANDBETWEEN(65,90)). Now you can use the *Advanced* tab to open the *Console* and view the output list as you press *Start Data* and *Stop Data*. This will transmit packets of data such as "5,P" with a comma as separator. After each output, *Excel* recalculates the spreadsheet so we get new values for CH1 and CH2. The trouble is that the micro:bit can't distinguish a number from a character and treats all incoming serial data as strings. So in order to write the receiving code in *MakeCode* we must get it to (a) read a whole 3 character line into a string, (b) store the first character in string CH1, (c) store the third character into string CH2, (d) convert CH1 into a number and display it, (e) have a rest, (f) display string CH2, (g) have a rest, and then repeat the process until told to stop by pressing button **B**.

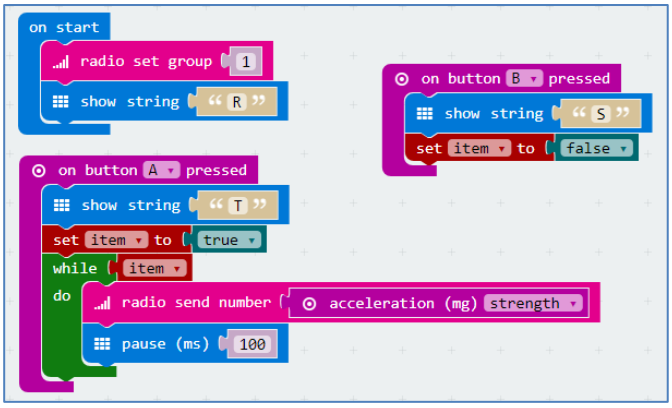




The brown string handling commands, such as *parse to integer*, come from the *Advanced*, *Text* menu. So now we have achieved two way communication between a micro:bit and a PC running *Excel* with the *Project Cordoba* plug-in. Coding the micro:bit for serial communication is very much easier than I had expected, based on the previous Arduino experience. If you don't have *Excel* or can't get hold of the *Cordoba* plug-in, the techniques will work fine with *Tera Term*.

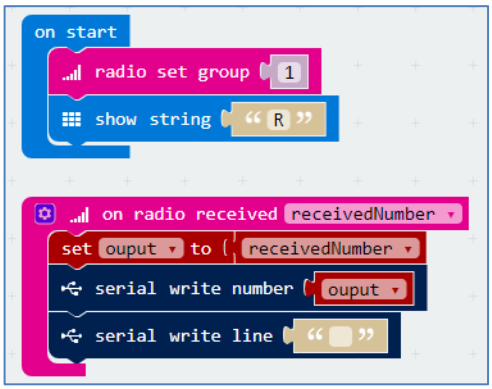
9. **Working in *Excel* without *Cordoba*** In April there was an interesting blog by Yidal Edery on Excel and Micro:Bit: Hacking for fun and creativity. This explains how to create your own macro in *Visual Basic* with *Excel*, and use it to create a dynamic data-logger. I very much hope that Microsoft

put *Project Cordoba* on open release very soon, as it is quite easy for even a beginner to use. But, for a more experienced developer than me, the VBA macro approach opens many doors in the meantime. Now we have the basic tools for two-way communication, we can start to develop some powerful applications connecting micro:bits to PCs. I will now try to build a simple data-telemetry system.

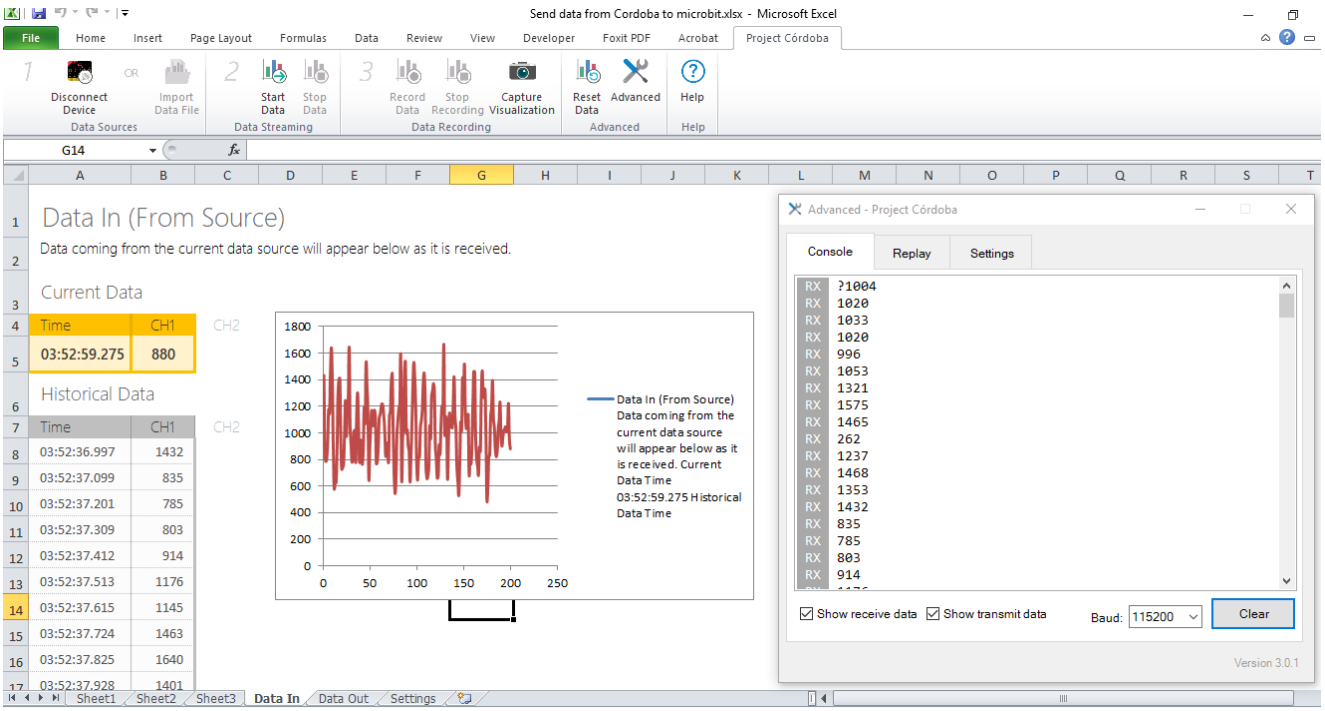


10. **Sending data from a micro:bit wirelessly to another, and passing it serially to *Excel***

Here I will try to send accelerometer data from micro:bit A using *Radio* commands to micro:bit B, which will then use *Serial* commands to send the data to *Excel* with *Cordoba*. It is unbelievably easy. The upper screen shot shows the sending program for the stand-alone micro:bit. The lower one shows the receive-and-send program for the micro:bit attached to the laptop. As we are only sending one channel of data, we can adjust the settings on *Excel*.

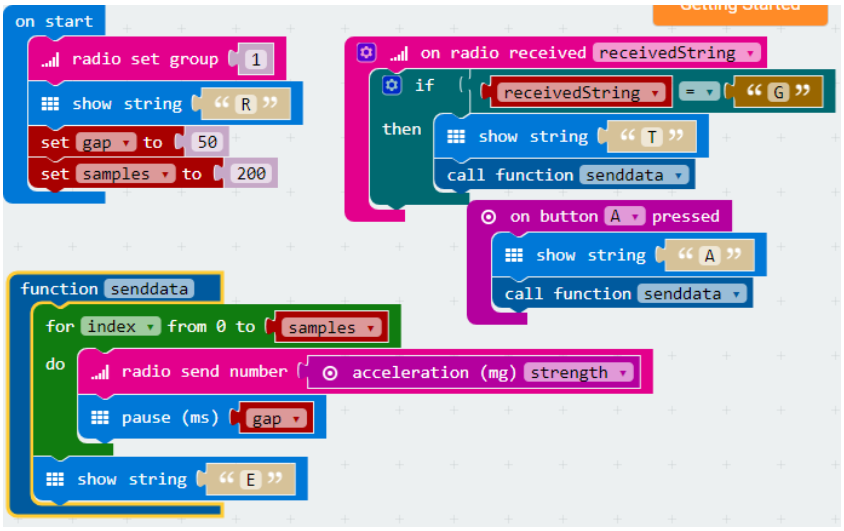


So that I can work on the stored data, I have set the number of data rows to 200. The received table of data can easily be graphed as shown in the screen capture.



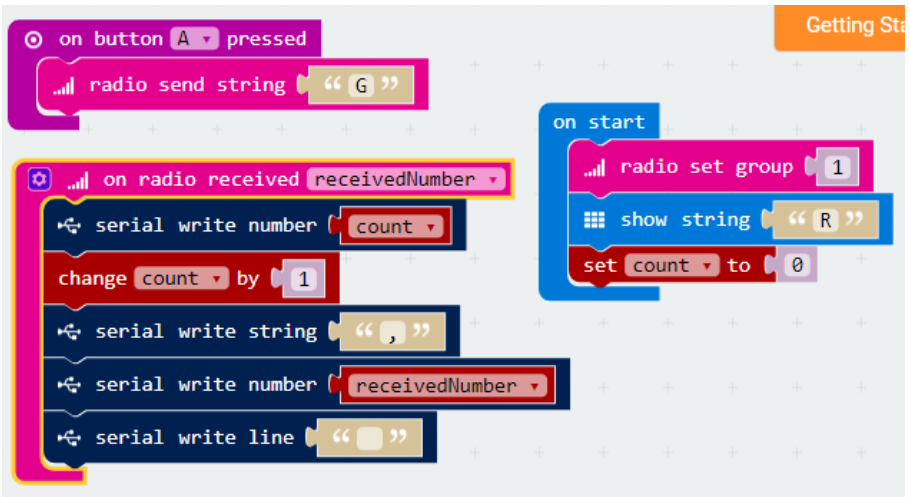
11. **Capturing simple harmonic motion with a micro:bit**

Now we have got the basics sorted out, we can go live with a dynamics experiment. I have strapped a micro:bit onto a heavy wooden elephant, called *Ellie*, who merrily bounces up and down on a spring.

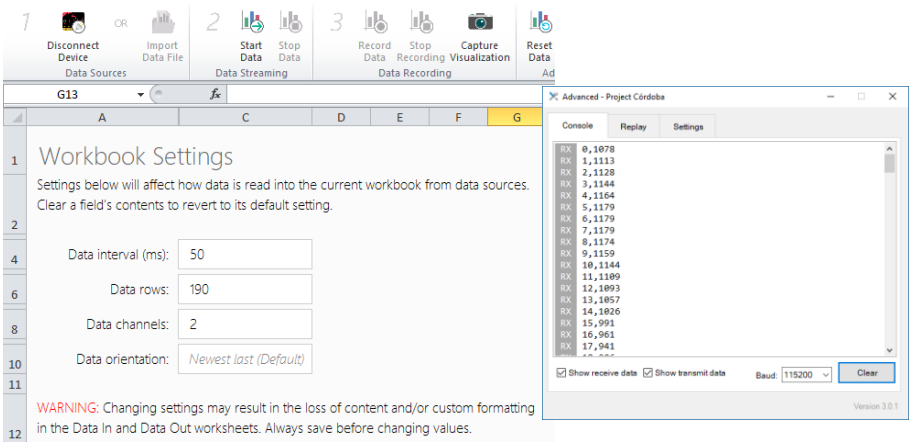


We just need some slight variations in each of the programs above. In the sending program we want data to start being logged when it receives a signal from the receiving micro:bit. So I have decided to use button **A** on the receiver to send a string to the transmitter which will trigger the data-logging. The program has four blocks. The `*on start*' block sets up the *Radio* group, displays `*R*' for `Ready' and sets the time between samples as 50 milliseconds, and the experiment length as 200 samples. I have kept an `*on button*' block so I can do a test run without having a second micro:bit set up. This, and the `*on radio*' block, both call the function `*senddata*' which does the data collection and transmission. The core of the function is a counted loop.

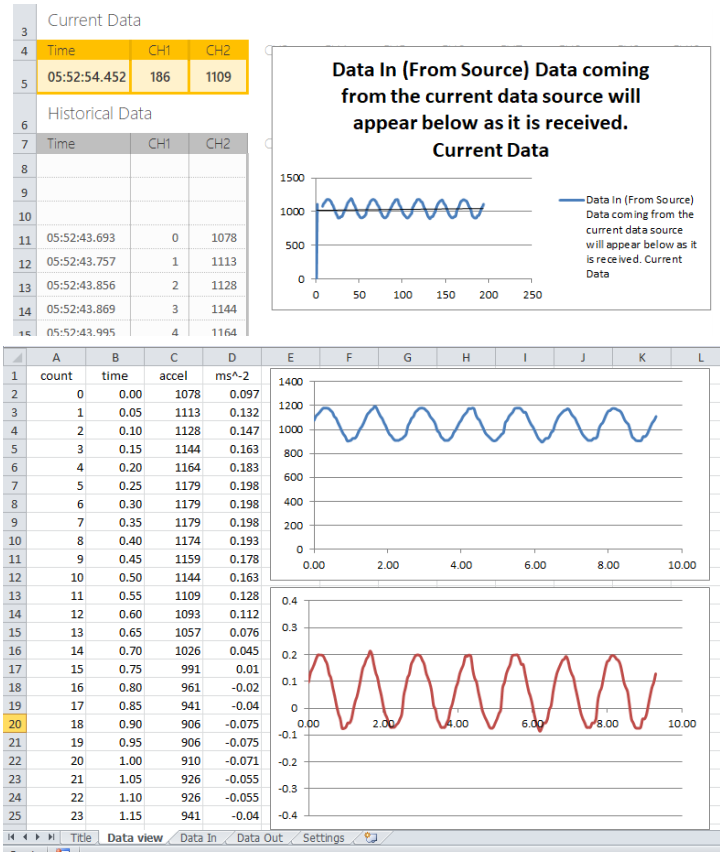
The receiving program has three blocks. The `*on start'* block sets up the radio, displays `*R*' for `Ready', and initialises the variable called `*count*'. The `*on button*' block just transmits string `*G*' for `Go' to the sending micro:bit. The `*on radio*' block writes 2 values, separated by a comma, into a line consisting of the value of the `*count*' variable and the received number which will be *Ellie*'s current acceleration. It also increments the `*count*' variable.



So once we have downloaded and flashed the programs to the respective micro:bits, we just have to set up the *Excel Cordoba* spreadsheet. In the `*Settings*' tab we can set the Data interval to 50 ms, the Data rows to around 200 and the Data channels to 2 (count and acceleration). Now, with

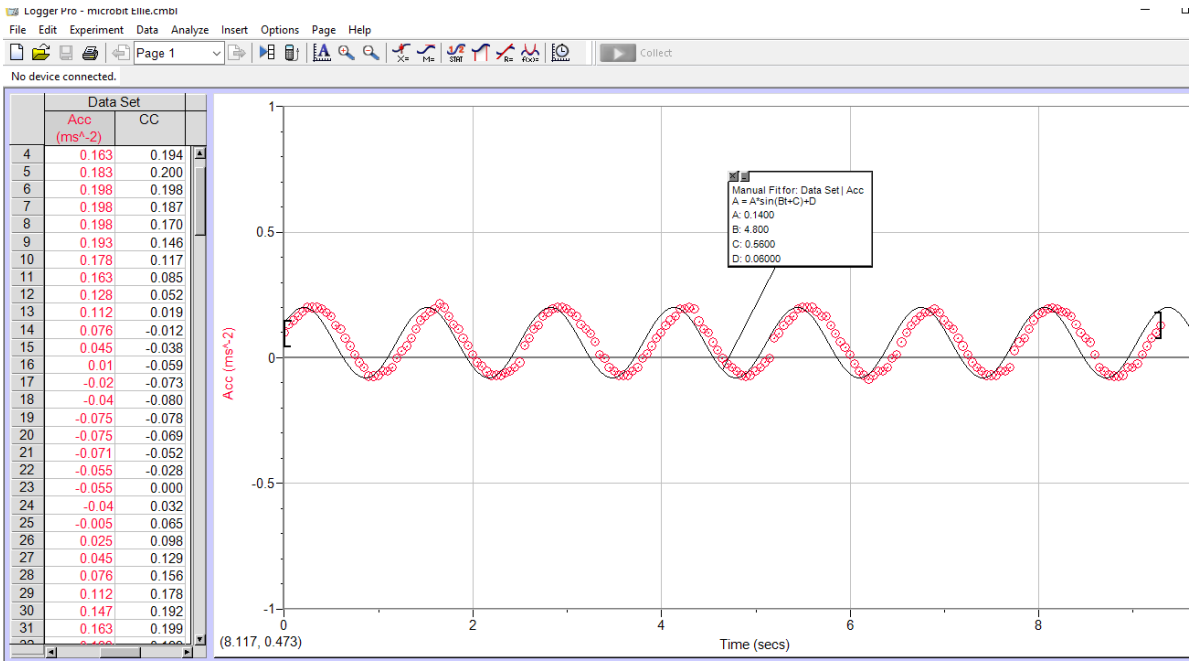


the receiving micro:bit connected, we can connect the micro:bit (mbed device) to COM 7 and use the *Advanced* console to check that the Baud rate is 115200 bits per second. So now is the time to set Ellie bouncing, and then get back to the laptop. Click on `*Start Data*' and then button **A**. With luck, the transmitting micro:bit now displays `T' and starts transmitting accelerometer data to the receiving micro:bit. This combines them with the `*count*' values and writes them as serial lines



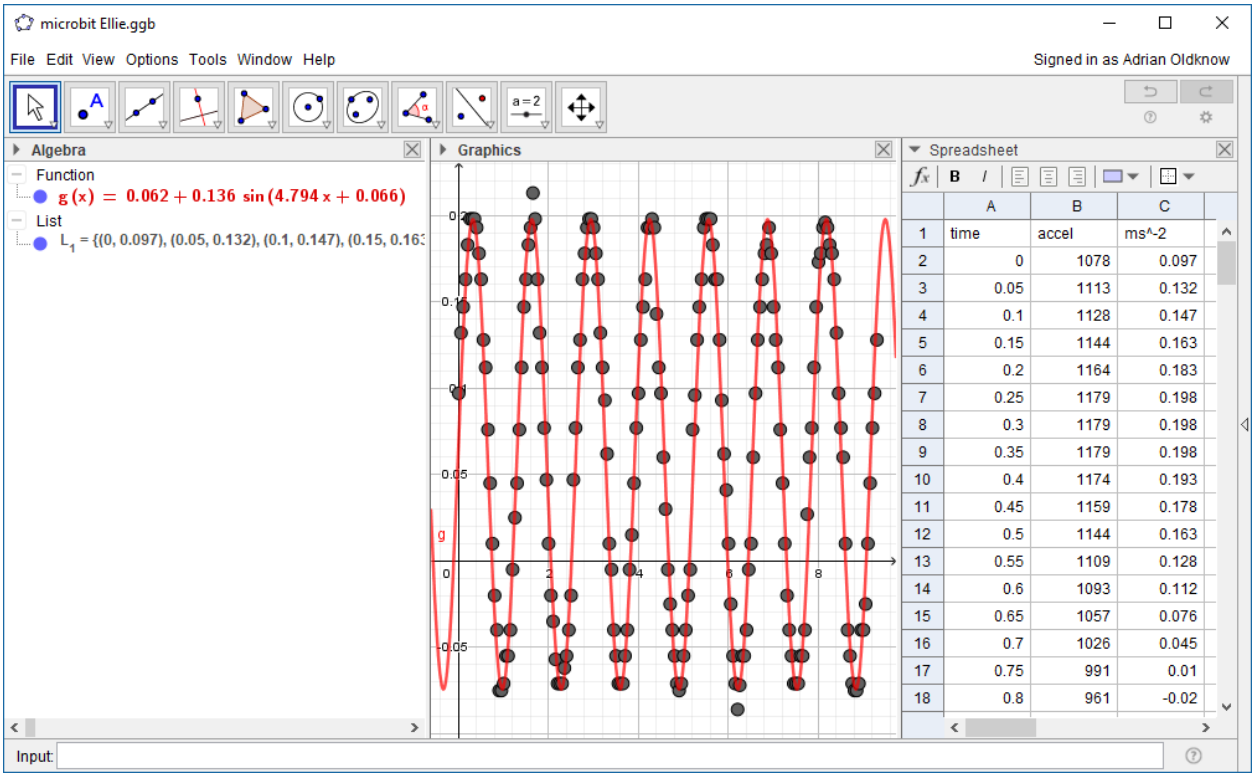
to *Excel*. As the data is received so it appears in the *Console*, if opened, and fills up the cells in the first 3 columns of the `*Data View*' sheet.

Once the data has been collected you click on `*Stop Data*'. Now you can play with them within *Excel* or copy and paste them to some other application, such a Vernier's *Logger Pro 3*, or the open-source, free *GeoGebra* software. I have renamed the second tab in the *Excel file* as `*Data view*'. The data from CH1 and CH2 have been copied and pasted into Columns A and C. Column B multiplies the `*count*' values in Column A by the `gap' value of 0.05 seconds (i.e. 50 ms). Column D divides the accelerations by 1000 and subtracts the acceleration due to gravity (9.81) to give *Ellie*'s actual accelerations in ms^{-2}. Below is an analysis carried out by copying and pasting data from *Excel* into Vernier's *Logger Pro 3*. I have selected to fit a *Model* to the data, and adjusted the parameters for a sine function fit by hand to make a manual fit. I could also have taken a video clip of *Ellie* in motion and synchronised it with the data.



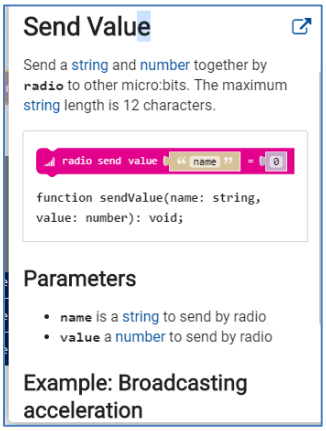
You can use *Tera Term* in place of *Excel* to read and store the stream of comma separated data to copy-and-paste into *Windows* applications, such as *Logger Pro 3* or *GeoGebra*. Vernier have their own Wireless Dynamic Sensor System which pairs and transmits data using Bluetooth. So it should be possible to use a micro:bit directly as a wireless sensor system for *Logger Pro 3*!

Here is a similar treatment of *Ellie*'s data using *GeoGebra*'s spreadsheet view. I have used a `*2-variable regression analysis'* and again fitted a sinusoidal model. Here *GeoGebra* has computed the best-fit parameters automatically.



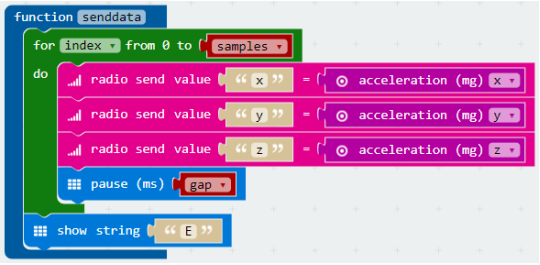
GeoGeobra developed its own *Sensors App* for mobile devices, which could transmit data from their built-in sensors to *GeoGeobra* on a PC using the Internet, and this was developed further by the Hungarian *Geomatech* project into the *Geomatech Sensors App*. Again it would be great if there was a simple Internet or Bluetooth connection which allowed data sensed by the micro:bit to be imported directly into *GeoGebra*.

12 Using *Radio* commands to send data packets One of my reasons for doing this work was to learn more about the *Serial* commands for the micro:bit. The *MakeCode* website does have *Reference Documentation* for each of the blocks, but this is not always helpful, or up-to-date! For example, the *Text* blocks aren't shown in the Reference Manual but are here if you search for them. I have used two of the *Radio* commands in the last example. We sent a data stream using the *radio send number* command, and a single string using the *radio send string* command. So I was curious to see if we could send a collection of possibly different types of data in the way we did by building data lines with the *Serial* commands. *Google* searches didn't reveal anything helpful for blocks, just for *Python*. So I decided to see if I could crack how to use the *radio send value* command.



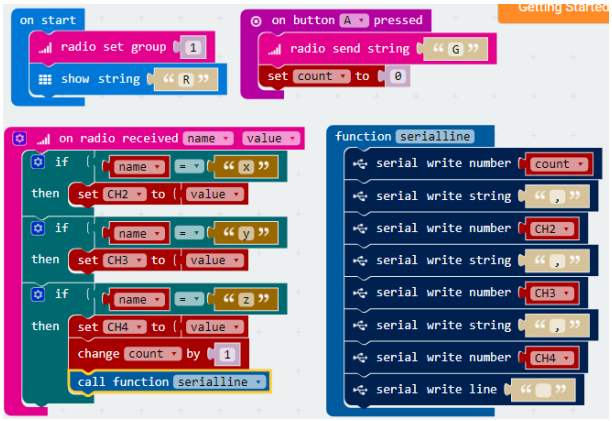
The worked example, to broadcast acceleration, wasn't very helpful, so I decided to try to send all 3 of the accelerations sensor readings separately. By giving them names such as `x', `y' and `z', I hoped that I could write a receiving program to build them up into a serial line to send to the PC.

So here is a simple modification of the Sending program. Actually all it does is to replace the single *radio send number* command in the *senddata* Function with three *radio send value* commands like:

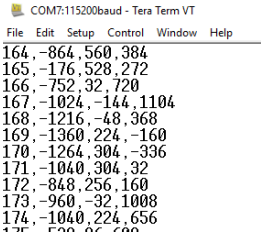


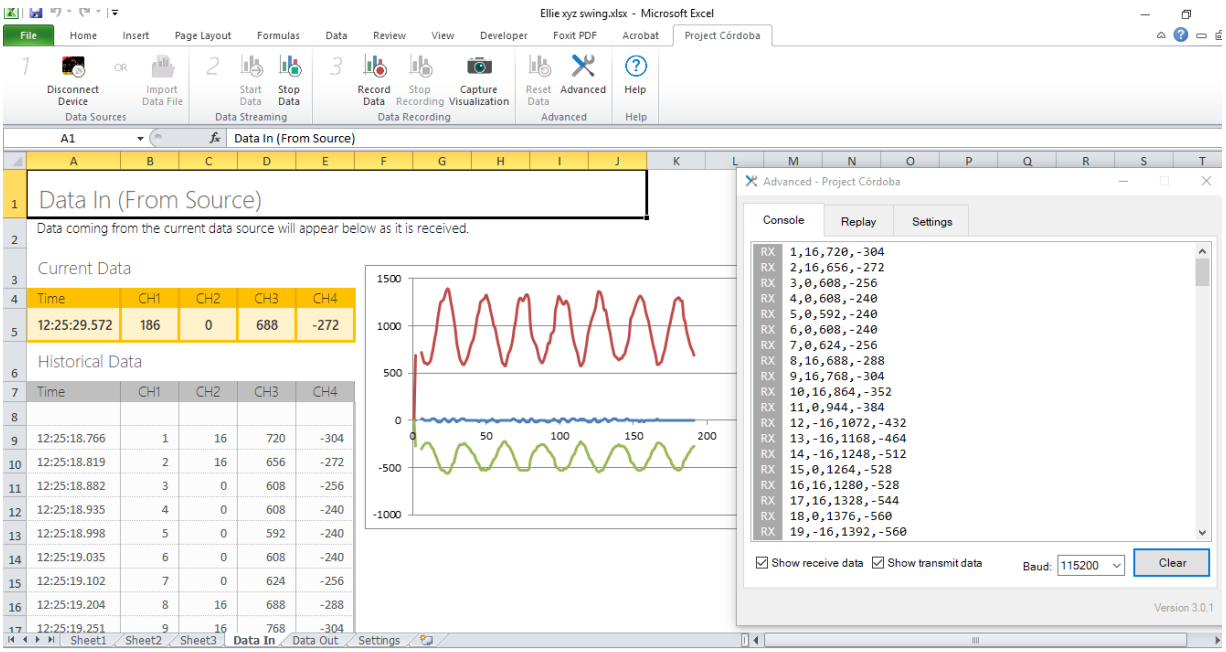
In the Transmitting program we will need to use the equivalent *on radio received name value* block:

I've split the Transmit program into four blocks. The `*on start*' and `*on button*' blocks are the same as before. The `*on radio received*' block has been extended using *Logic* blocks, and also calls a *Function* named `*serialline*'. The first two `*if*' commands store the *x* and *y* data values into variables *CH2* and *CH3*. The third does the same for *z* in *CH4*, but also



triggers the act of composing the serial line to send to the PC. It updates the *count* variable and then calls the `*serialline*' Function. That puts the 4 numbers together in a data line separated by commas. We can run a quick test using *Tera Term*. The we are all set to go live by setting *Ellie* on a more interesting ride swinging up and down and round and round!

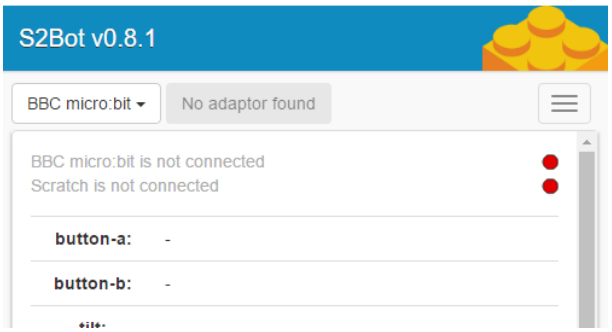


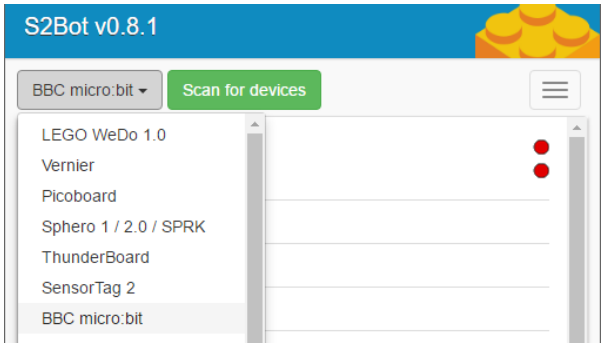


Much easier than I was expecting! So the `*radio send value'* and the `*on radio received name value'* combination give a powerful way to exchange data wirelessly between a pair of micro:bits and the PC. I will finish this article with a quick look at one way to use Bluetooth for the connection.

13 Using Bluetooth to connect micro:bit to *Scratch*

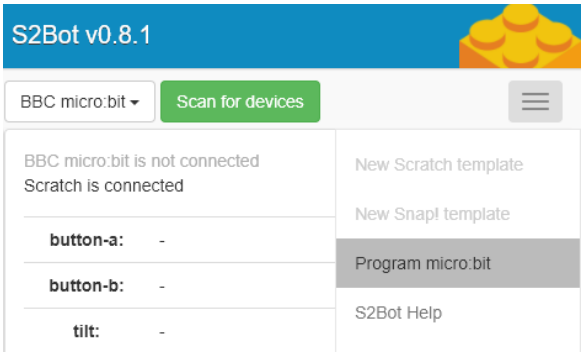
This was developed by Clive Seager of *Revolution Education* in Bath. The free App from the *PicAxe* site is called [*S2Bot*], and provides an interface for a wide range of robotic, and electronic, devices to exchange data with the offline version of *Scratch* running on a computer.

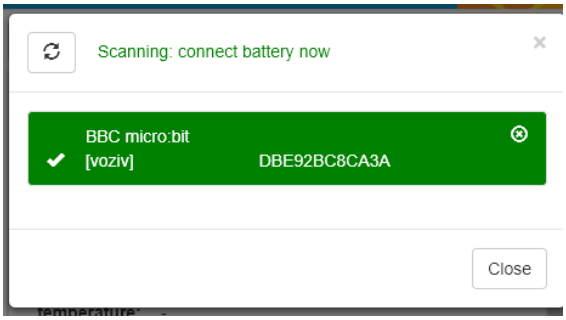


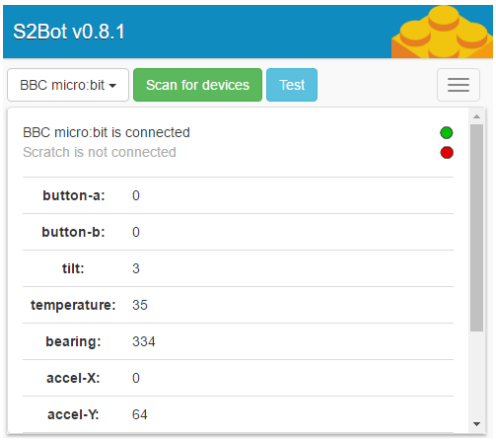


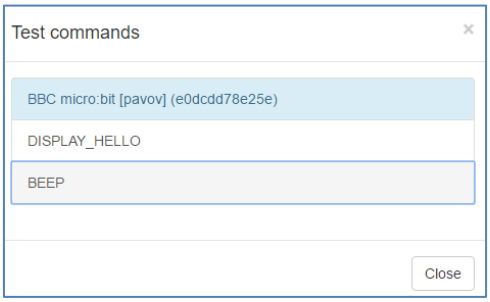
In order to connect effectively it is designed to be used in combination with a Bluetooth Low Energy (BLE) device such as the BLED112 bluetooth dongle, costing c£12. If your PC has Bluetooth built-in, you will need to turn this off first. When the dongle is plugged in, you can select `*BBC micro:bit*' from the drop down menu at the top left, and you should see the green `*Scan for devices*' tab appear. In order to transmit data over Bluetooth, the micro:bit has to

run a communications program. This is called *microbit-s2bot.hex*. It can be downloaded from the menu button at the top right of the *S2Bot* window. Select the `*Program micro:bit*' link and you will invited to download and save it to a folder on your PC. Once this has been done, you can connect your micro:bit with the USB cable and send the file to it. When it has finished flashing its led, you can disconnect the USB cable and connect a battery to the micro:bit. Micro:bit will now ask you to *'DRAW A CIRCLE*' to calibrate the sensors. Do this by tilting the micro:bit around until all the outer LEDs are lit. You should then see a smiley face to show that calibration worked. If instead you see a triangle of 3 dots disconnect the battery and then try the calibration again. Now you are ready to connect by pressing the green `*Scan for devices*' button on *S2Bot*. You should see the name of your micro:bit ready to connect. Click on the name and it should turn green, and the letter `*C*' (for `*connected*') will appear on the m:b. After a few seconds you will see some of the values sensed from the m:b appear on the App's display. Close the window and you should see that there is now a green dot showing that you have a BBC micro:bit connected. As well as seeing the sensor's readings you can also test the



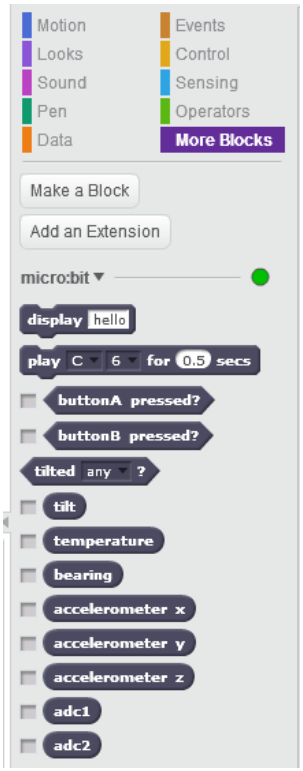


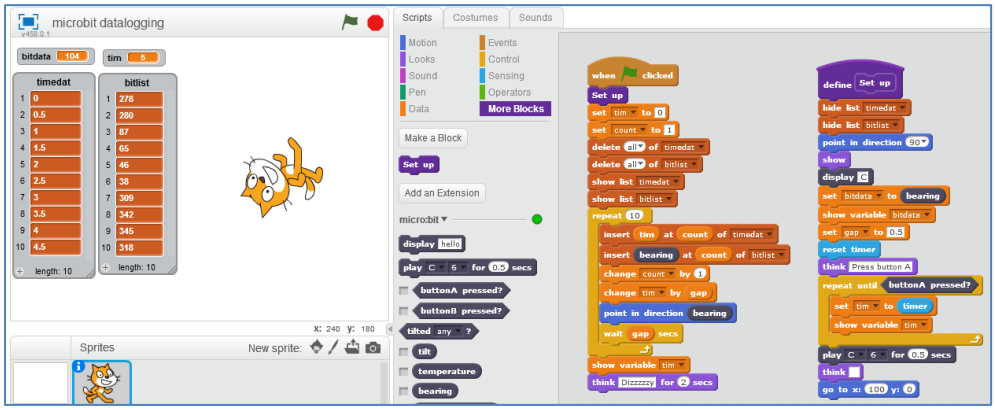




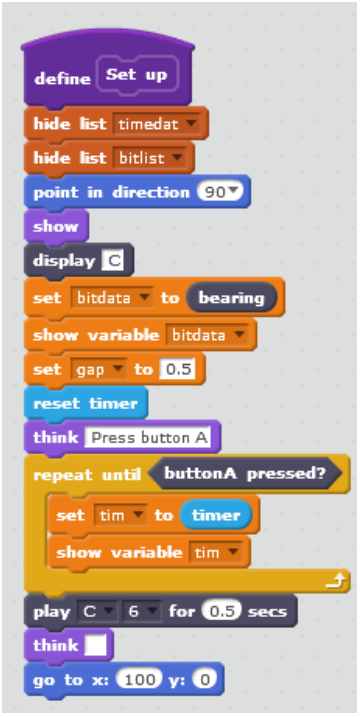
2-way connection by pressing the blue `*Test*' button. If your micro:bit is in a *Kitronik* M1-power case it will have a buzzer attached to pin P0, or you connect your own to make a `beep' – or just display `*Hello*' on the led array. Now you have connected a BBC micro:bit to the PC using a Bluetooth Low Energy transmission, and can see that it has 2-way communication. So in theory any application running on the PC should be able to use it connect with your micro:bit. I hope this could be possible with apps such as *Excel*, *Logger Pro 3* and *GeoGebra*. For now, though, we will show how *Scratch* uses it for 2-way communication. You need to install the offline *Scratch* editor which can be downloaded from here. You will need to install the Scratch template which adds the m:b command blocks. Use the menu to make the `*New Scratch template*'.

Store the resulting `*microbit_template.sb2*' file in a suitable directory. Now open the *Scratch* editor and use `*File*' to `*Open*' this file. Select `*More Blocks*' to see the extra commands now available in *Scratch* to work with the connected micro:bit. The first two are outputs from *Scratch* to the micro:bit's led display and sound output. The next two respond to the micro:bit's buttons **A** and **B**. The next 7 respond to some of the micro:bit's on-board sensors (not light nor magnet field). The last two read analog values from pins P1 and P2. But you can't write to them (yet). So here is an example program to log some compass data and to use it to control *Scratch*'s display.



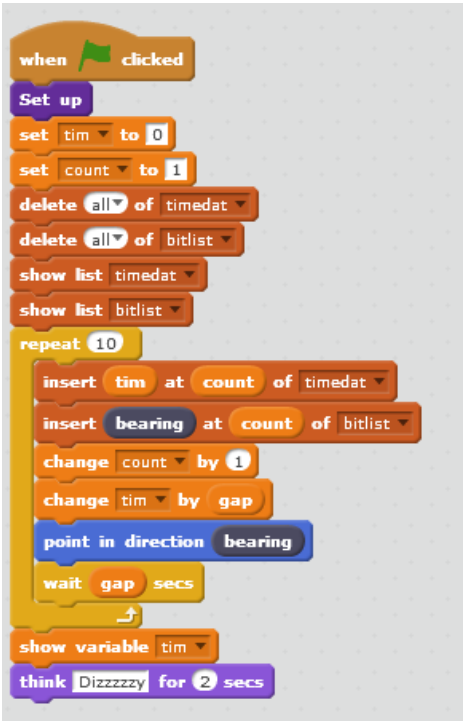


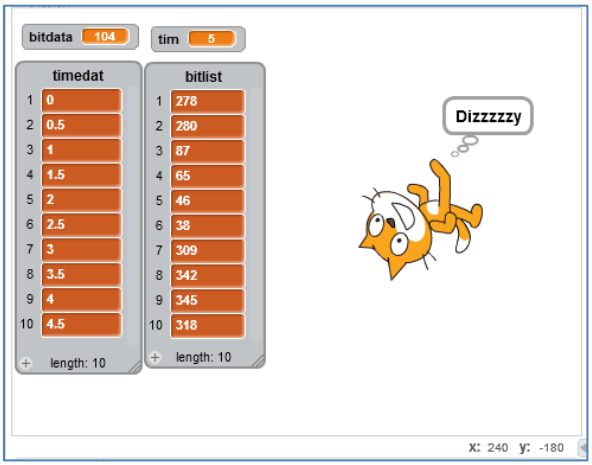
The first command calls the `*Set up*' block. Let's start there. We use the `*Data*', `*Make List*' command to create two lists called `*timedat*' and `*bitlist*', which will hold the times and sensed data for the experiment. We need to hide these and make *Scratch* point right. Then we send a `*C*' to the micro:bit to show we're clear to start. We have created a variable `*bitdata*' to hold the micro:bit's bearing and display it; also the variable `*gap*' to set the interval between readings. We reset the clock and display a message to press button **A** on the m:b to start the collection. The loop just shows the time going by, until button **A** is pressed. Then we play a sound on the micro:bit and remove the message, before moving *Scratch* to the right half of the screen. Now we are ready to start storing values into the two lists, as well as using the `*bearing*' value to make *Scratch* point in different directions.



(Note. I have been unable to make S2Bot work properly since the most recent update to *Windows* 10. The micro:bit displays `*C*' and `*D*' when S2Bot opens and closes the connections, but no data actually gets received. I hope someone can find the cause and share the solution. It worked fine on the previous version of *Windows*.)

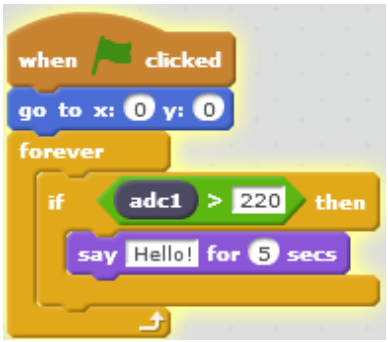
Now back to the main program. When the green flag is clicked we want to call the `*Set up*' block just created above. We create two new data variables, `*tim*' and `*count*', and give them initial values. We want each run of the program to create fresh data lists, so we delete any previously stored data, and then display to empty lists. For this demonstration we only have a short counted loop to create 10 sets of readings. We write the current value of the `*tim*' variable into the current line of the `*timedat*' list, and the current value of the m:b's `*bearing*' into the corresponding line of the `*bitlist*'. Then we increase the `*count*' variable by 1 ready to write to the next line, and increase the `*tim*' variable by `*gap*'. Then we use the m:b's `*bearing*' to control the direction Scratch is heading. We have a pause of length `*gap*' seconds before repeating the process. Finally we display the value of the elapsed `*tim*', and make *Scratch* display a message. Remember to save your work!

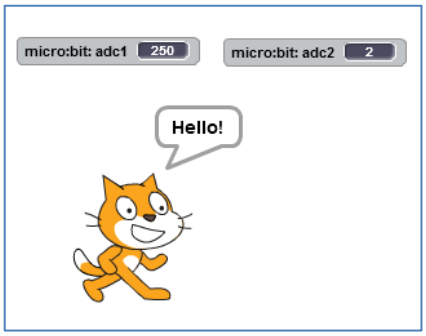




Now we have seen the mechanics of writing *Scratch* programs accepting micro:bit sensor input, the world is, as they say, your oyster. If you want to add your own sensors on pins P1 or P2, then *Scratch* will read these as *adc1* and *adc2* (for `*analog to digital convertor*'.) My final example uses pin P1 with a variable resistor (potentiometer) to make something happen at a threshold value.

I have put ticks by the micro:bit Data variables `*adc1*' and `*adc2*' so that *Scratch* displays their current vales. Pin P2 is not connected, so the value stays constant at roughly zero. Trimming the potentiometer attached to pin P1 returns values varying between around 200 and 250. So I have chosen my threshold value as 220. The little program just detects when the incoming voltage is greater than the threshold and makes *Scratch* display a message. There are lots of ways which you can build simple circuits and add additional sensors to use the m:b to control *Scratch* objects.

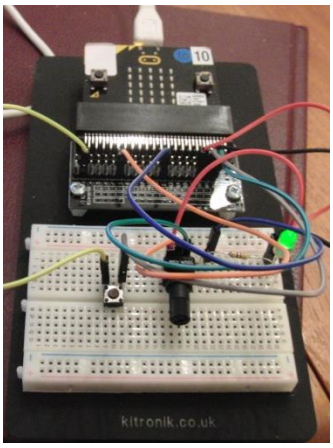




14. Back to BASICs I began this journey by referring to the BASIC programming language. So that's where I'll finish. There is a free version of BASIC for *Windows* which has support via serial cable for the BBC micro:bit. It is called *FUZE Basic* and can found here. This will communicate with a number of devices such as Raspberry Pi and Arduino. On the *Fuze* web page there is the link to download the hex file you will need to install on your micro:bit. There is also a link to the reference manual where pages 249-251 explain the commands relating to the BBC micro:bit. Here you can both read and write to the micro:bit's pins. You can find out more in sections 8 and 9 of my `*First steps in Computing with the BBC micro:bit*'.

```
REM Control an LED
LOOP
mbANALOGWrite(2,0)
WAIT(0.5)
mbANALOGWrite(2,150)
WAIT(0.5)
REPEAT UNTIL mbButtonA
END
```

```
REM First flash an LED
LOOP
    mbANALOGWrite(2,0)
    WAIT(0.5)
    mbANALOGWrite(2,150)
    WAIT(0.5)
REPEAT UNTIL mbButtonA
REM Now dim the LED
LOOP
    pot = mbANALOGRead(1)
    mbANALOGWrite(2,pot)
    WAIT(0.5)
    PRINT pot
    PRINT
REPEAT UNTIL mbButtonB
END
```



15. Afterthoughts I deliberately set out to explore how to set up and use data connections between micro:bits and PCs by just using the currently available blocks in the MS *MakeCode* editor. I continue to be amazed at the power and versatility of this combination across the spectrum of teaching , learning and using STEM subjects in schools. *MakeCode* is free and the micro:bit, at c£15, costs less than a round of drinks for the family at the local pub! (Of course many secondary schools also have stocks of undistributed micro:bits they received in Summer 2016.) In my *First steps in robotics with a BBC micro:bit*, I set out explore whether this combination would open up simple cross-curricular micro:bit robotics projects to Primary Schools at Key Stage 2. Now, with the capacity to work with existing software such as *Excel*, *GeoGebra*, *Logger Pro 3*, *Scratch* and BASIC, I am also convinced we have ideal tools to support their use across STEM subjects including Computing, Design Technology, Mathematics and Science. Coupled with imaginative learning resources such as `micro:bit in Wonderland', these extend to other aspects of the curriculum such as Art & Design, English, Geography, History, Sports and Wellbeing. Free tools, such as Microsoft's *Project Cordoba*, PicAxe's *S2Bot* and the *Bitty DataLogger*, are paving the way to making the micro:bit very easy to use in many powerful ways. I very much hope these will continue to be developed, and that others will follow.

Of course, the blocks approach to programming is only one way to bridge the technologies. Teachers, students and supporters of education have the combination of skills with *Python*, *Mbed*, *C++*, *VBA* etc to extend the versatility, almost *ad infinitum*. What I hope most sincerely is that we can combine forces so that those with the skills to develop the tools put them to good use for the benefit of learners and teachers across the STEM, and other, subjects and across the age range.