# Author:

I am Colin Grant, a freelance Java developer from Berkhamsted, Hertfordshire. After taking part in the PiWars 2019 competition I have started tinkering with micro controllers such as the micro:bit, Raspberry Pi and Arduino.

# Notes:

All pictures embedded in this document are also available as jpg or png images in the root folder.

Whilst screenshots have been provided of the code, the full source code for micro:bit (microbit-SerialTutorial.hex) and Raspberry Pi (SerialTutorial.py) have been supplied as you may have better ways of displaying this.

They are also available on the public git hub repository: <https://github.com/colin-grant/serial-tutorial.git>

# Main Article Text:

*A circuit board

Description automatically generated<title\_picture.jpg>*

*micro:bit connected to Raspberry Pi with USB cable*

micro:bit and Raspberry Pi Serial communications over USB

# Background

In 2019 my family and I entered the annual Pi Wars competition in Cambridge with a robot called Sputnik. Although the main controller had to be a Raspberry Pi, we also wanted a way of displaying which operating mode the robot was in, and also have a way of changing modes on the robot in case we had issues with remote control on the day.

This is where the micro:bit came in. It has a simple display for showing the operating mode of the robot - and 2 buttons to allow us to change modes. Also, I hoped to be able to use the compass on the micro:bit to see which direction the robot was facing.

So the next issue was how to communicate between the Raspberry Pi and the micro:bit. The simplest answer was to use the USB port for serial communications. The same cable could also be used to power the micro:bit from the Raspberry Pi.

This article shows how you can achieve simple communication between a Raspberry Pi and a micro:bit using a USB cable.

# What you need

* micro:bit
* Raspberry Pi with Python 3 installed
* micro USB cable to connect the micro:bit to the Raspberry Pi

# Objectives

We’re going to develop a simple project that shows how to set up serial communications between the micro:bit and a Raspberry Pi using the USB cable. We’ll create a simple protocol for sending the data and look at some of the pitfalls that you may come across.

In this project, the ‘A’ and ‘B’ buttons on the micro:bit will be used to request the time and date respectively from the Raspberry Pi. When the time or date are received they’ll be displayed on the LED display.

The Raspberry Pi will also request the current compass heading periodically.

# The Protocol

When communicating between any 2 processors, they both need to be speaking the same language in order to know what the individual messages mean. This is the communications protocol.

For this project we’re going to have a simple protocol that consists of a 3 character message type and optional value, separated by a colon character.

So if the when the micro:bit want’s to ask the raspberry pi for the time it sends a message “TIM:” and the Raspberry Pi will respond with a message “TIM:11:34”.

We’ll use the message types:

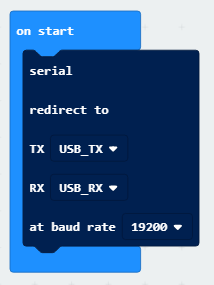
* TIM – request and response for current time
* DAT – request and response for current date
* CMP – request and response for current compass heading.

We can easily add message types for other data values – such as the micro:bit accelerometer values.

# The micro:bit code

## Step 1

The Serial blocks are in the ‘Advanced’ section of the make code editor. On startup we need to tell the micro:bit to send all the serial data to the USB port using an ‘on start’ block.



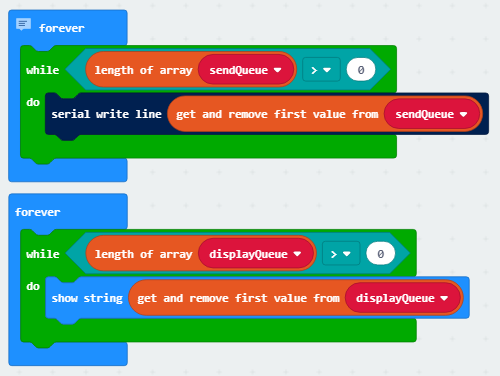
*<makecode\_on\_start.png>*

*makecode block showing serial setup.*

This also sets the baud rate to 19200. This is the rate at which bits are sent down the USB cable. This can be any setting but must be the same as that used on the Raspberry Pi.

## Step 2

Create forever loops to handle serial writes and message displays. These use list variables as queues. The serial write task ensures that we’re only trying to output messages to the serial device one at a time so they don’t get corrupted. The message display task ensures that we can display a message in the background, and then go back immediately to reading messages from the serial device.



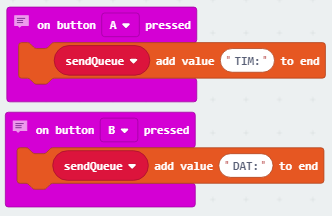
*<makecode\_queue\_handlers.png>*

*makecode block showing send and display queue handlers.*

The ‘serial write line’ block will write any messages on the send queue to the serial device.

## Step 3

When the A button or B button are pressed, we add a message to request the time or date to the serial send queue so that the background task created in Step 2 will send it to the serial device.

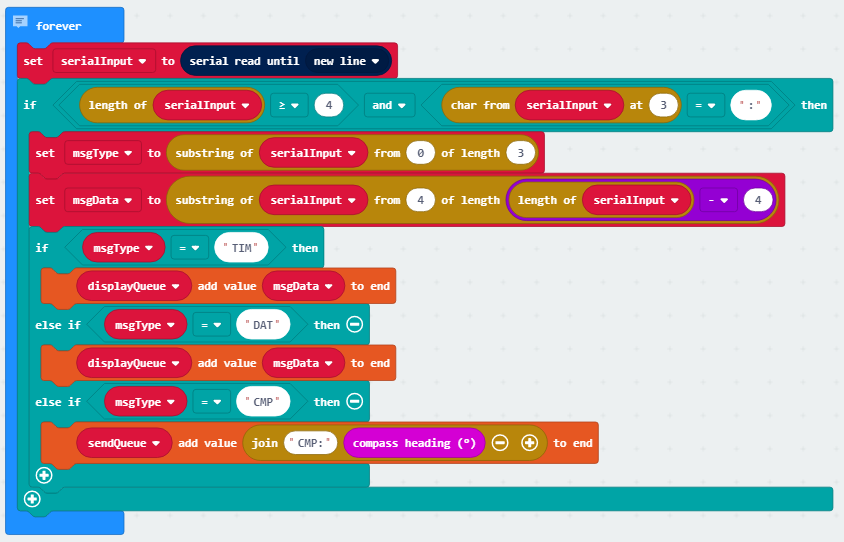


*<makecode\_button\_handlers.png>*

*makecode block showing button A and button B handlers.*

## Step 4

Now we add another forever background task to read messages from the serial device.



*<makecode\_button\_handlers.png>*

*makecode block showing button A and button B handlers.*

‘serial read until new line’ will wait for a message terminated by a new line character to be received.

We then check that the message is long enough and has a colon in the right place. It is possible for messages to get corrupted, so this gives us some confidence it’s a valid message.

The substring blocks extract the message type and the message data from the message.

We then decide what to do depending on the message type. For ‘TIM’ and ‘DAT’ messages we add the data to the display queue so that the time or date can be displayed. (This is where the queue comes into its own – we don’t need to wait for the message to be displayed before receiving the next serial message).

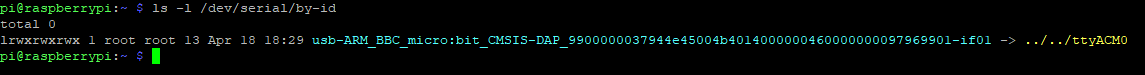
For ‘CMP’ requests, we construct a message with the current compass heading and put it on the send queue to be sent to the Raspberry Pi.

Any other message types we ignore.

## The Raspberry Pi Code

Now that the micro:bit is all set up we can move to the Raspberry Pi.

When you plug a serial device into one of the USB ports on the Raspberry Pi, it creates a device file link in the /dev/serial/by-id folder.



*<pi\_serial\_device\_file.png>*

*Screenshot of ls showing serial device file when micro:bit connected.*

Using the pySerial python library, we can read and write messages using this link. For Sputnik, we went through the list of links in this folder and created a thread to read from each one, since we also communicated with an Arduino using the same code. However, for simplicity we’ll assume here that we only have a single micro:bit connected, and we’ll handle both the reading and writing in the same thread.

One thing to note is that the serial python library works with the bytes datatype rather than strings. The micro:bit uses ASCII strings, so we need to encode and decode using the ‘ascii’ character set when converting.

So here is the code in full:



*<pi\_full\_python\_code.png>*

*Screenshot of python code on raspberry pi – source code also included as <SerialTutorial.py>.*

### Lines 1-5

These are the imports we need for the os, time, datetime and serial libraries.

### Lines 11-18

Here we find all the device files in the /dev/serial/by-id folder and use the first one we find to create a Serial object. We need to use the same baudrate we set on the micro:bit. The timeout value states we will wait 200ms for a message to be received before timing out any serial read operations.

### Line 22

Inside a forever loop, we read a line from the serial device. This may timeout, allowing the rest of the code to run, rather than waiting constantly for some data.

### Lines 24-27

We check to see if we have a valid message. If so we break the message into the message type and message data parts. Note the use of b’:’[0] when testing for the colon character. This is required since there is no other way of creating a single literal byte in python.

### Lines 29-37

If we receive a request for time or date, we use the datetime library to format the date or time and then write it to the serial device. Note that we encode using ‘ascii’ to convert the string into a bytes data type compatible with the micro:bit.

### Lines 39-40

If we receive a compass heading, we output the compass heading in a formatted string. Again we decode using the ‘ascii’ character set.

### Lines 42-45

This is the code that sends a request for the current compass setting. Each time we request the heading, we reset the next poll time to 2 seconds in the future so that we send a request every 2 seconds.

### Lines 46-52

If the micro:bit is not plugged in when we start the python program then we display a warning and exit immediately.

## Running the code

Install the code onto the micro:bit and connect to the Raspberry Pi using a USB cable.

Run the python code on the Pi and you should see the compass heading displayed on the console output every 2 seconds. Press one of the buttons and the date or time should show up on the micro:bit.