Using the radio

Programming for beginners with the BBC micro:bit

John Davies

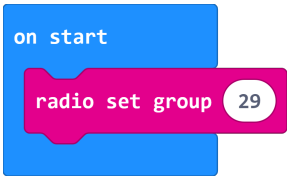
2026 May 13

1 Introduction

Introduction

The **radio** in the micro:bit can send packets (short messages) to another micro:bit.

An obvious question is: how do you know which micro:bit(s) will receive the message? This controlled by setting a **group** in the radio, which you should do at the start. All radios in the same group will receive a message but others will not.



This is something like tuning the frequency on an old-fashioned radio. The group can go from 0 to 255 and must be the same for the transmitter and receiver.

2 Simple use of the radio

Sending and receiving

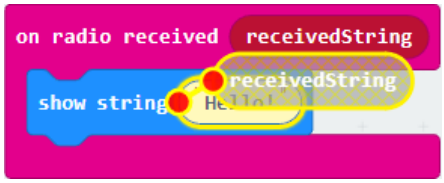
The simplest blocks send or receive a number or a string (short text):

radio send number	—	on radio received *receivedNumber*
radio send string	—	on radio received *receivedString*

Here is how they are used to send a simple text message.

You need to program two micro:bits to test this, set to the same group. The send block on one activates the receive block on the other.

To use the variable receivedString, drag it out of the on radio received block. This creates a copy, which you can then drop in the show string block.



Now extend the program to send a more personal message.

Other possibilities include a teleporting duck, which appears to send an image from one micro:bit to another when the board is shaken, and an indoor–outdoor thermometer (and logger).

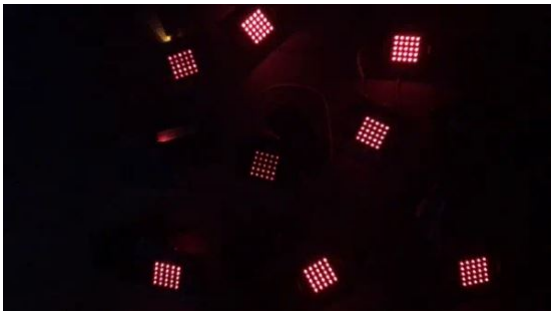
You could put only the send block on one micro:bit and the receive block on the other, in which case a message can be sent only one way. But it's easier to download the same program, with both blocks, onto the two micro:bits so that you can send in either direction. The on radio received block puts the message that it receives into the variable receivedString, which you can copy into the show string block.

Another pair of functions communicates both a name and value. Further details are on the reference page for the radio.

3 Fireflies

Fireflies program

An amusing, if useless, application of the radio is to simulate a swarm of fireflies, which flash more-or-less in synchrony. This example is simplified from Fireflies on the micro:bit web site, due to Nicholas H. Tollervey.

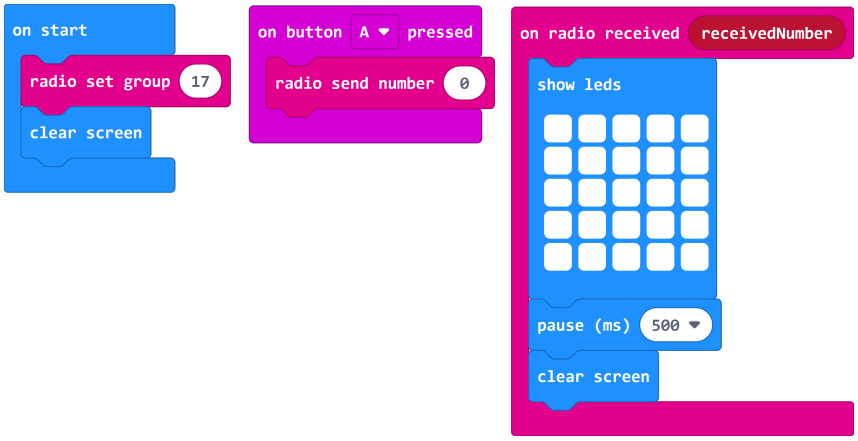


The basic idea is that one firefly sends a radio signal that causes the others to flash.

I'll go through a sequence of programs, adding features each time.

Fireflies 1

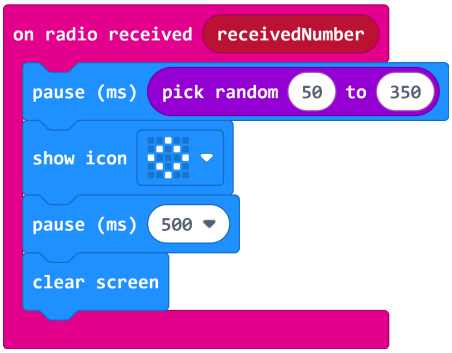
Here is a starting point, which simply flashes all LEDs for $\frac{1}{2}$ second when it receives a radio signal. (The value of receivedNumber is ignored.)



We could instead send the duration of the flash in the radio message and use the value of receivedNumber in the pause (ms) block.

Fireflies 2

More realistically, the fireflies do not flash immediately but have a reaction time. This might vary between individuals so add a random delay before the flash.



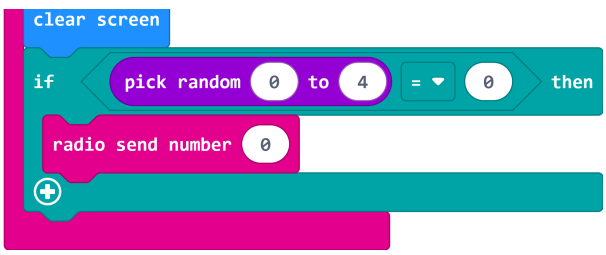
(I have changed show leds to show icon to sace space on the screen.)

Fireflies 3

At present, each firefly flashes only once after button A on one has been pressed. To keep the display going, a firefly should send a radio signal to trigger another flash from the others.

However, we don't want every firefly to do this every time because the display would become confused and would never end. Instead, only a small fraction of the fireflies should send a new radio signal after each flash.

This addition does the trick.



How does this work?

- The pick random 0 to 4 function is equally likely to produce any one of the five numbers 0, 1, 2, 3 and 4.
- The if–then block calls radio send number only if the random number is 0.
- Because 0 is only one of five, equally probable random numbers, a radio signal is sent after only one flash in five *on average*.

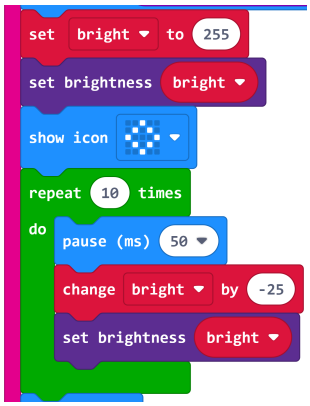
Sometimes only one firefly will send a signal but on other occasions more than one will transmit. Eventually none will do so and the sequence of flashes ends.

I chose the number 4 at random (!) but it should be roughly the same as the number of micro:bits taking part in the activity to get several rounds of flashing before they die out.

I should have used 2 or even 1 instead of 4 for testing the program on a computer in MakeCode, where only two fireflies take part.

Fireflies 4

At present, each firefly shines at full intensity for $\frac{1}{2}$ second (500 ms) and switches off abruptly. Perhaps it would be more realistic if the light faded gradually. The following blocks go between the random pause and clearing the screen.



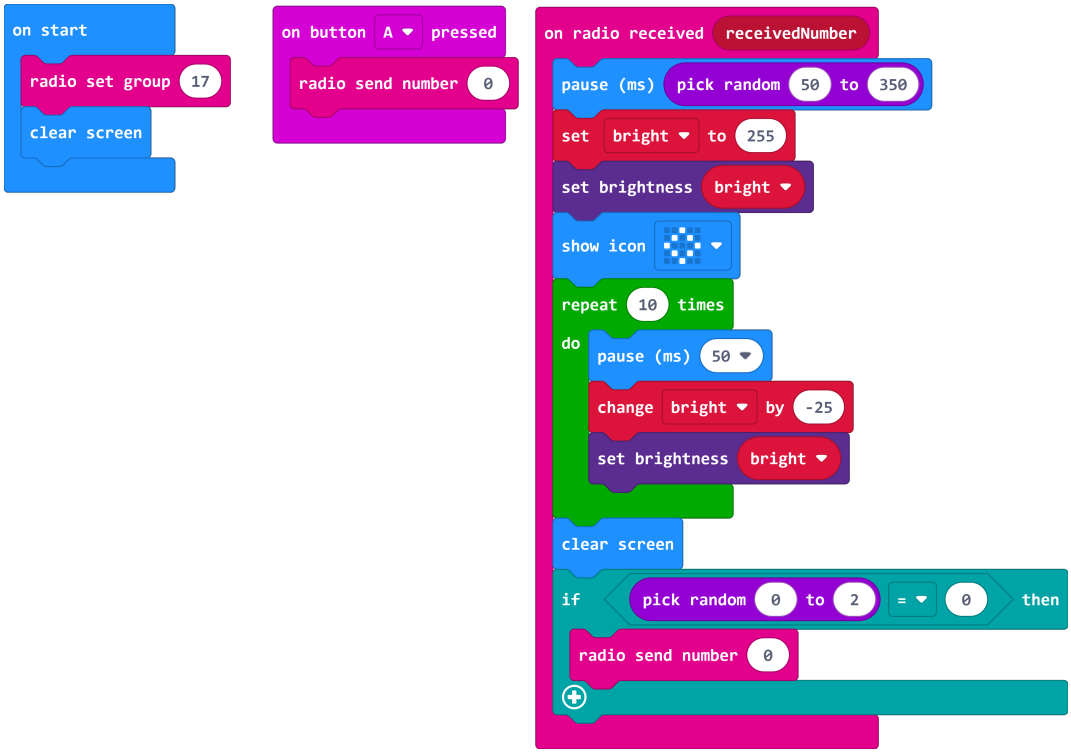
How does this work?

- The set brightness function is in the Led section under . . . More. It controls the brightness of the LEDs, as you might expect. The number (its argument) goes from 0 (off) to 255 (maximum brightness).
- You need to create the variable bright.
- The repeat loop executes its contents a set number of times, 10 here.
- Each time the loop is executed (each *iteration* is the jargon) the brightness is reduced by 25. We first change the variable, then use the variable in the set brightness block to dim the LEDs.

  After 10 iterations, the brightness has been reduced to $255 - 10 \times 25 = 5$, so the LED is very dim.

- Remember that variables must always be initialised (be given a value) before they are used! I have done this before the repeat loop.

Here is the complete program for reference.



The version on the web site is slightly more complicated and uses a function, which is good practice but which I have not yet explained.

Fireflies complete

That is nearly the version of the Fireflies program on the micro:bit web site. Here are some possible extensions.

- Make a more exciting display than the fading flash.
- Reduce the power of the radio transmitter so that only nearby fireflies respond to a request for a flash. You can find radio set transmit power in . . . more under the Radio section. Its argument goes from 0 to 7. A variable received packet signal strength in the Radio section can be used to test the range of the transmitter.
- Another project on the MakeCode web site simulates fireflies in a rather different way.