The micro:bit Bluetooth troubleshooting guide

Version: 1.3

Author: Martin Woolley

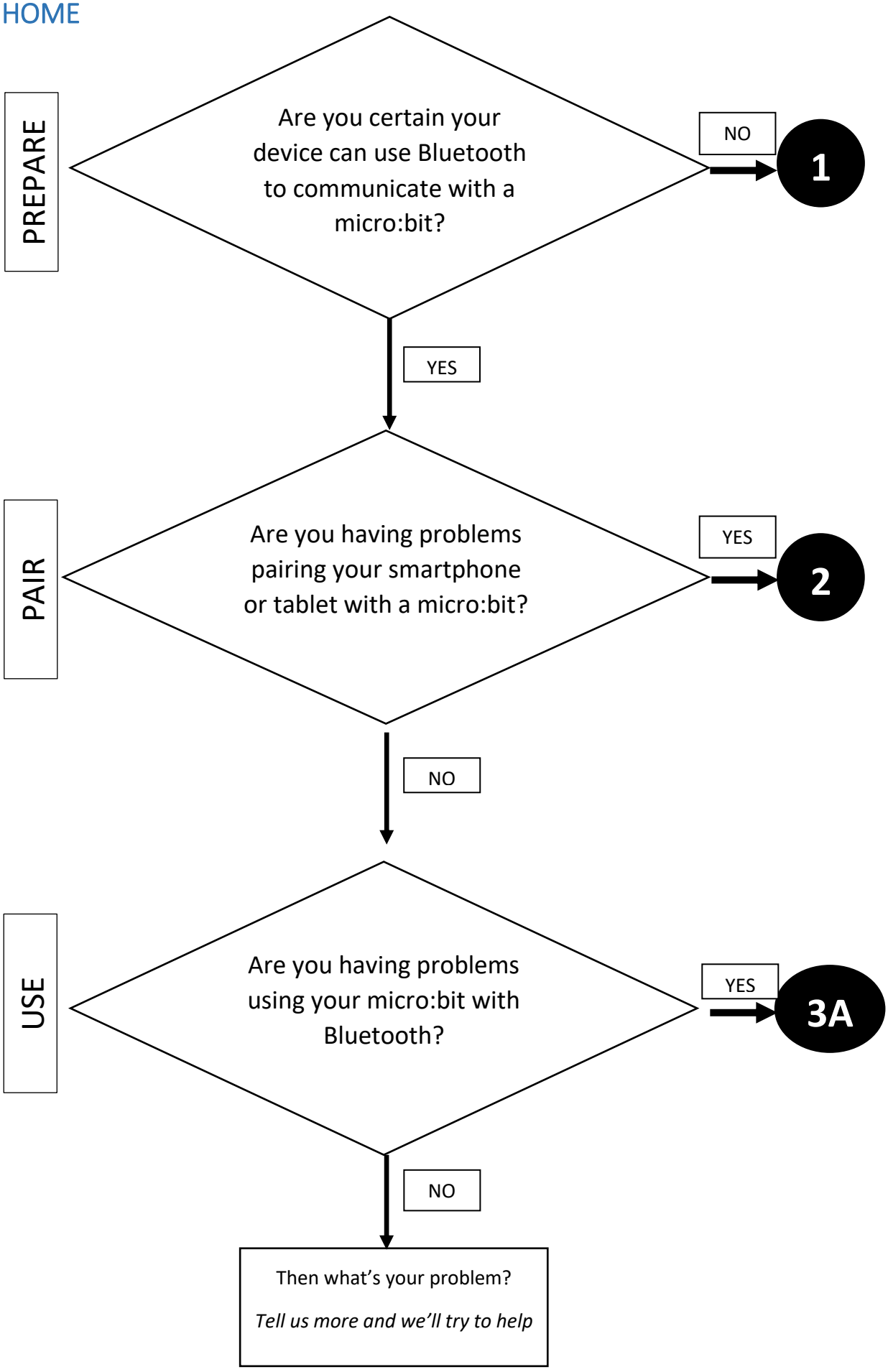
http://www.bittysoftware.com

Table of Contents

Change History

Version	Date	Author	Details
1.0	31/08/2017	Martin Woolley	Initial version
1.1	12/09/2017	Martin Woolley	Clarified that comments about the impact on pairing of flashing new code only applies to flashing over USB. Updated MakeCode pairing options details following requested change having been implemented by Microsoft.
1.2	18/09/2017	Martin Woolley	Added section on using mbed IDE. Added section relating to the BBC legacy editors, Blocks, Touch Develop and the CodeKingdom editor.
1.3	26/09/2019	Martin Woolley	Added point about the Android Location permission being required for scanning to work. Added point that a device connected to a micro:bit will prevent it from advertising and hence prevent it from being discovered by a scanning app.

HOME


PREPARE
Are you certain your device can use Bluetooth to communicate with a micro:bit?
NO
1
YES
PAIR
Are you having problems pairing your smartphone or tablet with a micro:bit?
YES
2
NO
USE
Are you having problems using your micro:bit with Bluetooth?
YES
3A
NO
Then what's your problem?
Tell us more and we'll try to help

1. Device Requirements

Smartphones and Tablets

For a smartphone or tablet to be able to use Bluetooth to communicate with a BBC micro:bit it must support **Bluetooth Low Energy**.

Android devices must be running at least version 4.4. Version 5.0 or later is recommended however.

iOS devices released since 2011 have Bluetooth Low Energy.

Types of Bluetooth

There are two different types of Bluetooth. The BBC micro:bit supports one of them but not the other. The type it supports is called Bluetooth Low Energy. Consult the documentation for your smartphone or tablet to check whether or not it supports Bluetooth Low Energy. These days, 99% of all smartphones and tablets on the market support this type of Bluetooth, which has been around since 2010, so unless your device is quite old, the chances are it does support Bluetooth Low Energy.

Note that Bluetooth Low Energy masquerades under various other names sometimes. For example:

Bluetooth Smart

BLE

Bluetooth 4.0

The first two are alternative terms which mean Bluetooth Low Energy. Bluetooth 4.0 means your device *may* support Bluetooth Low Energy.

Bluetooth Capabilities or "Roles"

Bluetooth Low Energy devices fall into several categories, according to the way they can communicate with other Bluetooth Low Energy devices. The BBC micro:bit is what's known technically as a "Peripheral".

Peripherals

A Peripheral can broadcast data to indicate to other devices that it is in range and available for use. This is called "advertising". A Peripheral can be connected to by another, suitable Bluetooth Low Energy device, known as a "Central" device. A Peripheral can only have one other device connected to it at a time. Once connected to, the Peripheral stops advertising and Central devices can exchange data over the connection in either direction.

Centrals

A Central device can search for Peripherals by listening for Bluetooth radio advertising broadcasts. This is called scanning. A Central device can initiate a connection to a Peripheral which is advertising.

Other Roles

For completeness, note that a Bluetooth Low Energy device may advertise but not accept connections (known as a Broadcaster) or scan for advertising packets but never attempt to connect (known as an Observer).

A BBC micro:bit is a Bluetooth Peripheral and may be discovered and connected to by a Bluetooth Central device like a smartphone. See http://bluetooth-mdw.blogspot.co.uk/2016/07/microbit-and-bluetooth-roles.html for more on this subject.

“Radio” is not Bluetooth

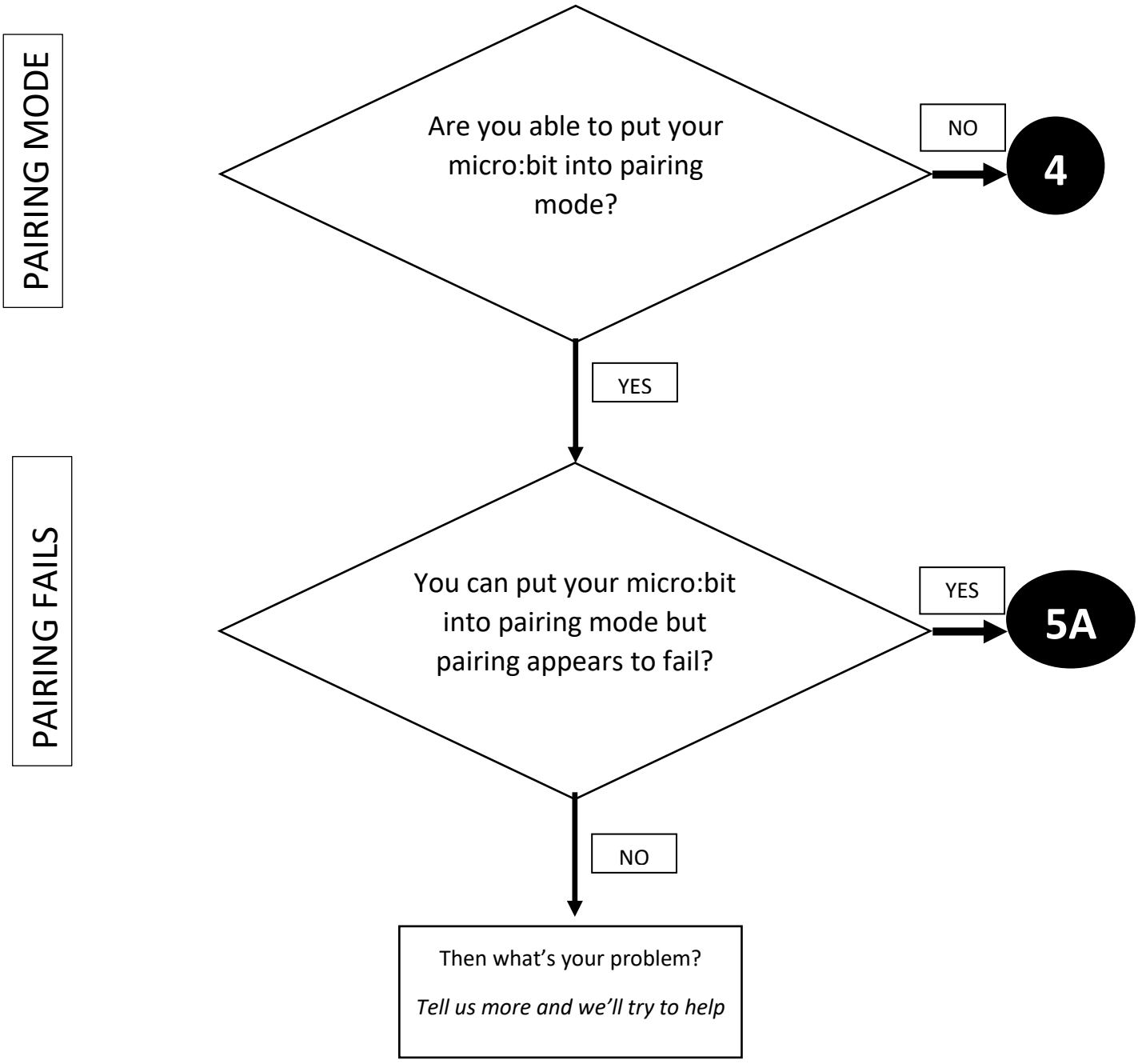
The BBC micro:bit has a wireless communications capability known as “Radio”. This is *not* Bluetooth and cannot be used to communicate with a Bluetooth device like a smartphone. Smartphones do not support the micro:bit “radio” capability.

Other Types of Device

The BBC micro:bit can be used with any other Bluetooth Low Energy “Central” device, as described above. Examples include smartphones and tablets or a Raspberry Pi (with Bluetooth hardware and software).

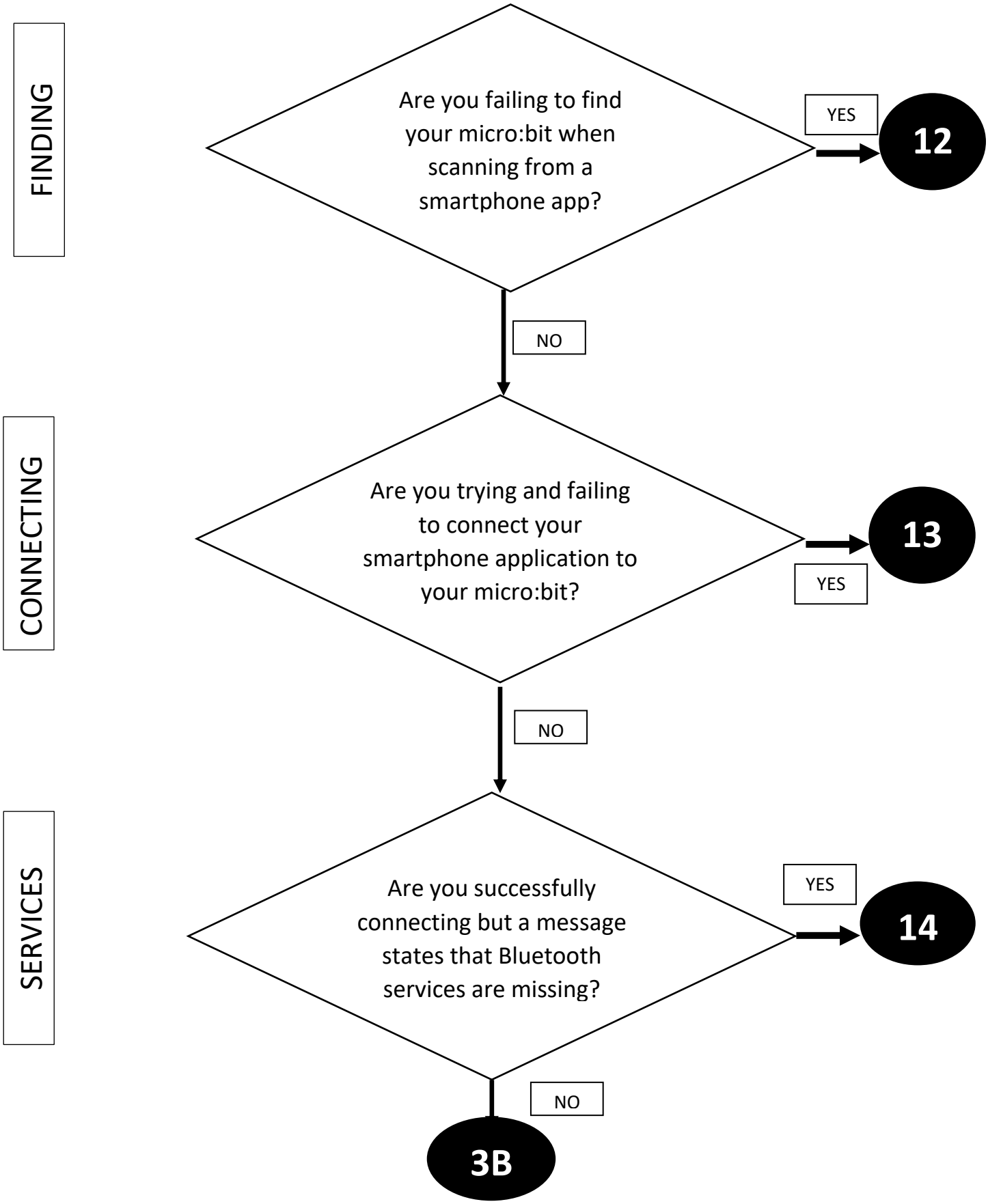
Two BBC micro:bits cannot be paired or communicate using Bluetooth without a special hex file, which is not available from the commonly used development tools. See section 17.

2. Pairing Problems



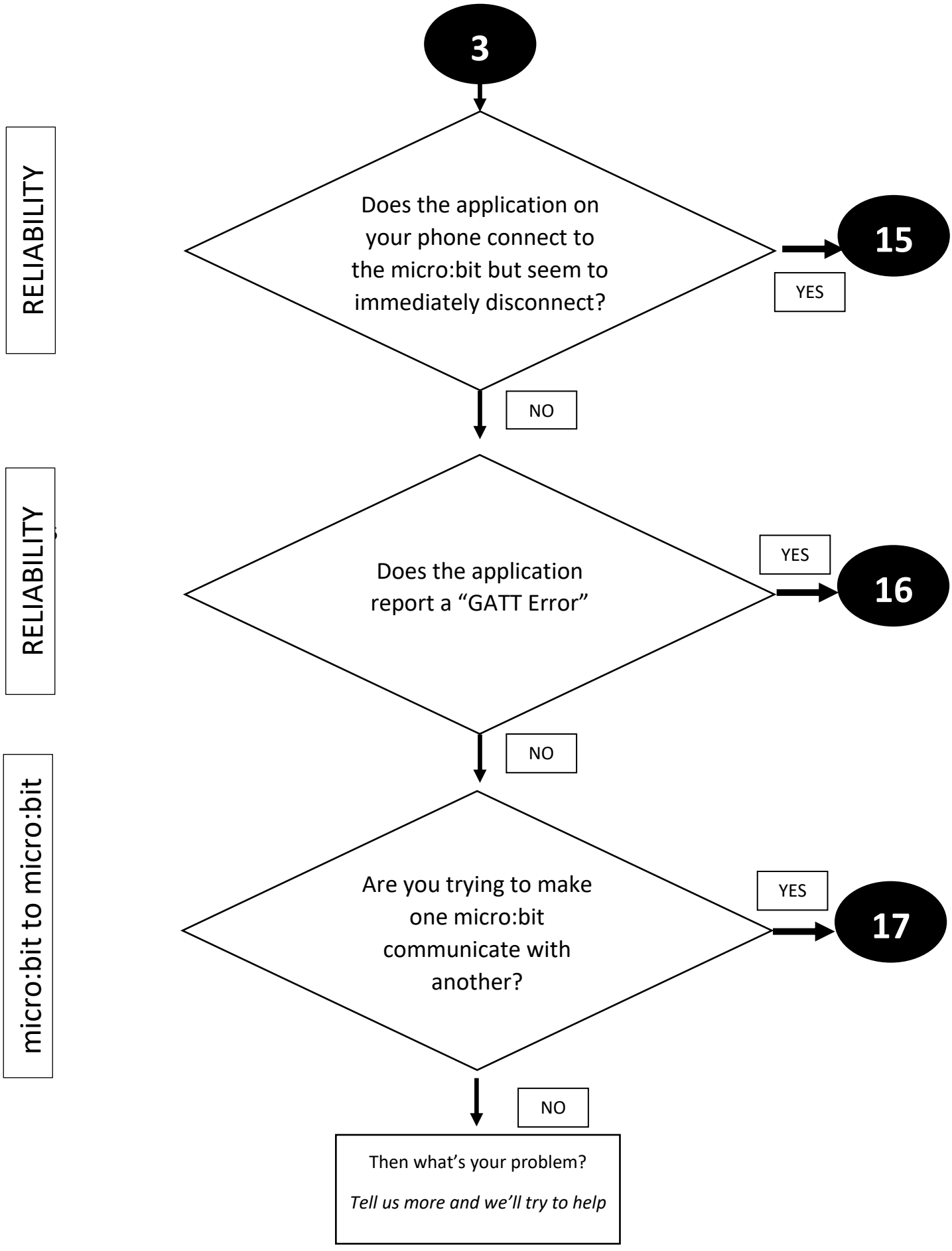
3A

3A. Problems using Bluetooth with a micro:bit

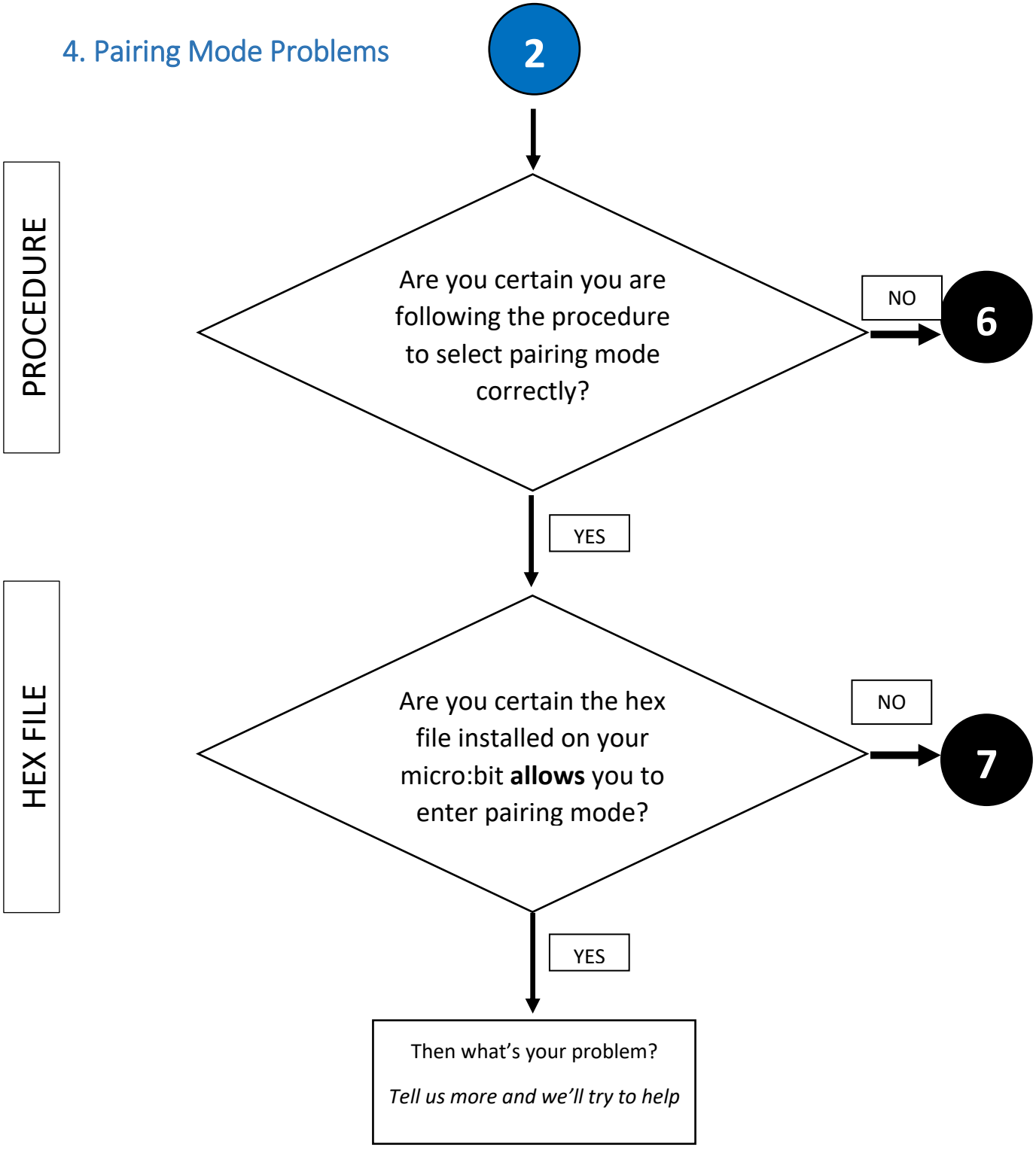


3B

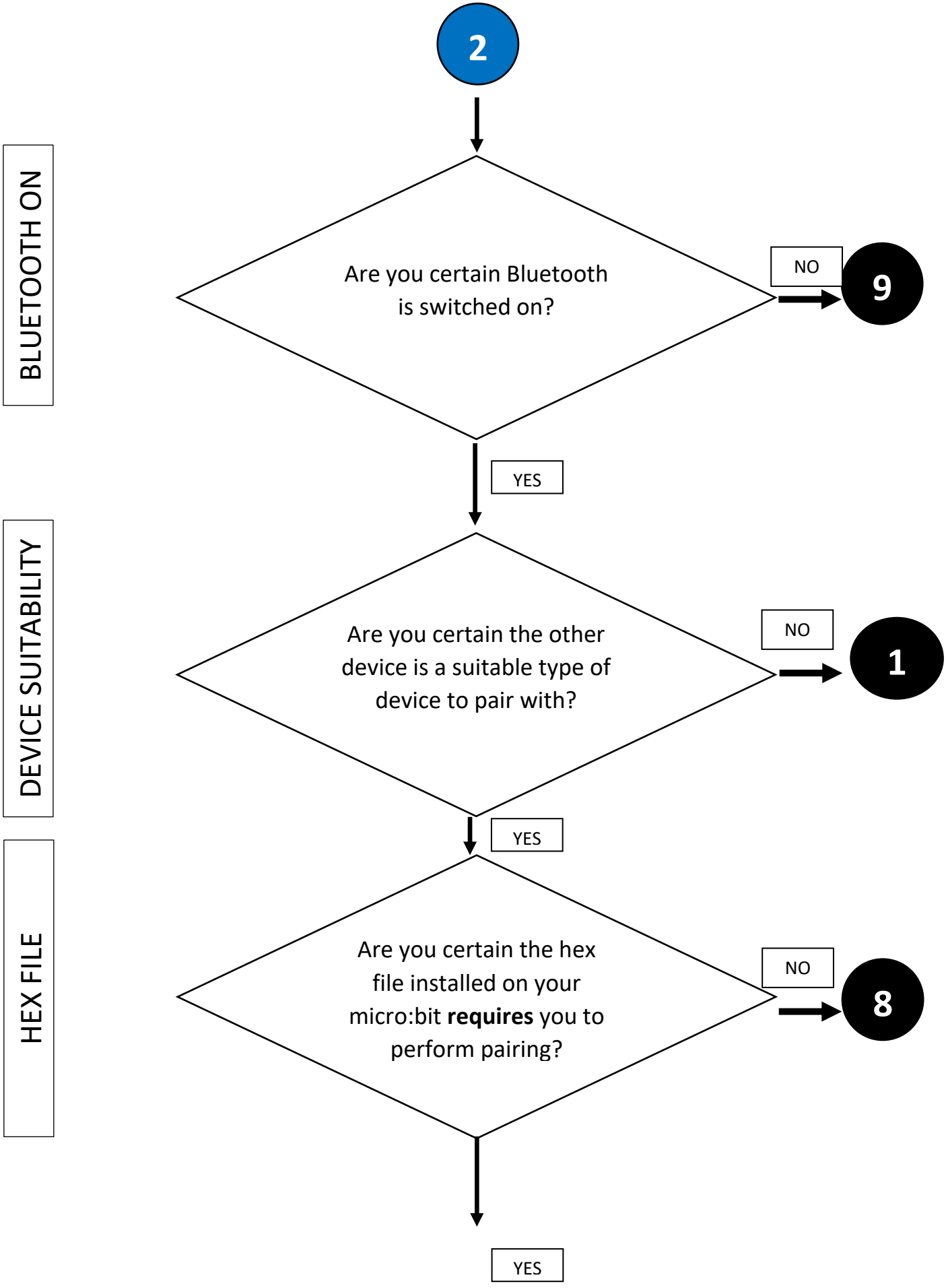
3B. Problems using Bluetooth with a micro:bit



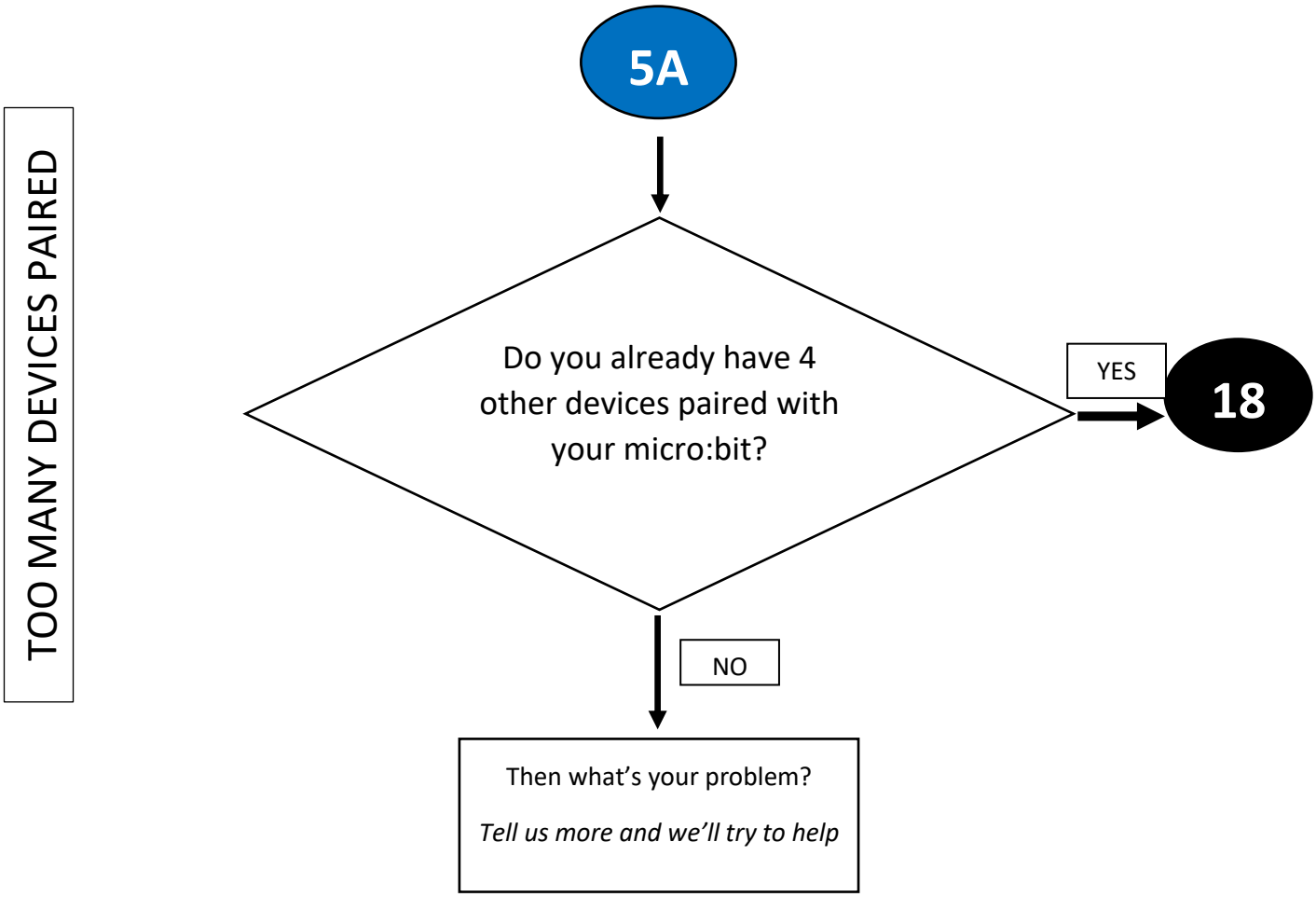
4. Pairing Mode Problems



5A. Pairing Fails



5B. Pairing Fails



6. The Pairing Procedure

Bluetooth pairing creates a trusted relationship between two devices and allows them to exchange encrypted data. It also ensures that only a device which has been paired with your micro:bit can connect to it. micro:bits only allow one device to connect to them at a time, so without this protection, any smartphone or tablet could connect to your micro:bit, possibly without you knowing and this would prevent you from connecting. Pairing adds some steps and inconvenience to using your micro:bit, particularly since you need to pair every time you flash a new hex file over USB to the micro:bit (since flashing over USB clears the Bluetooth pairing keys) but it does provide security. So if you're creating hex files or selecting between hex files which require pairing and those which do not, you need to think about whether security or ease of use is your priority.

Pairing is only sometimes needed. To use your micro:bit with another device over Bluetooth, you must pair with the other device if and only if the hex file on your micro:bit makes this necessary. See sections 7 and 8 for more on this.

If pairing is required, there are two variations to the pairing procedure, each a variation of how Bluetooth can accomplish pairing. The most secure variant is called **Passkey Pairing** and a less secure but easier to use approach is called **Just Works Pairing**. Which of these is used is, once again, determined by the hex file (see section 8).

The procedure for Android and iOS also differs.

In all cases, to pair your micro:bit with another, compatible device, requires you to clear any previous pairing details from your smartphone and then switch your micro:bit into a special mode called Pairing Mode.

Android

If you have an Apple device, go to the iOS section below. The procedure for Android and iOS devices is different.

1. In the Bluetooth --> Settings screen, check to see if your micro:bit is listed under Paired Devices. If it is, select the cog icon next to it and then select Forget. This will clear the pairing details from your phone so you can start the process afresh. A phone can be paired with more than one micro:bit at a time but if you need to repair because (say) you installed a new hex file and cleared the pairing data from the micro:bit, you *must* forget the corresponding pairing data on the phone and start again.

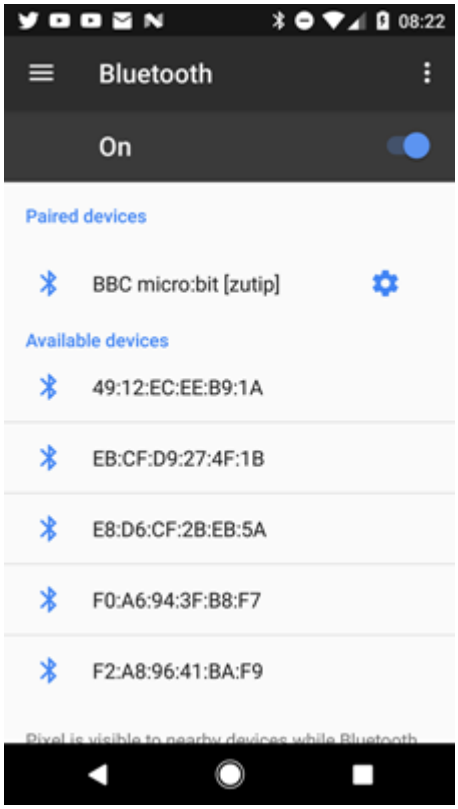


Figure 1 - phone is already paired with this micro:bit so we must clear the pairing before we proceed

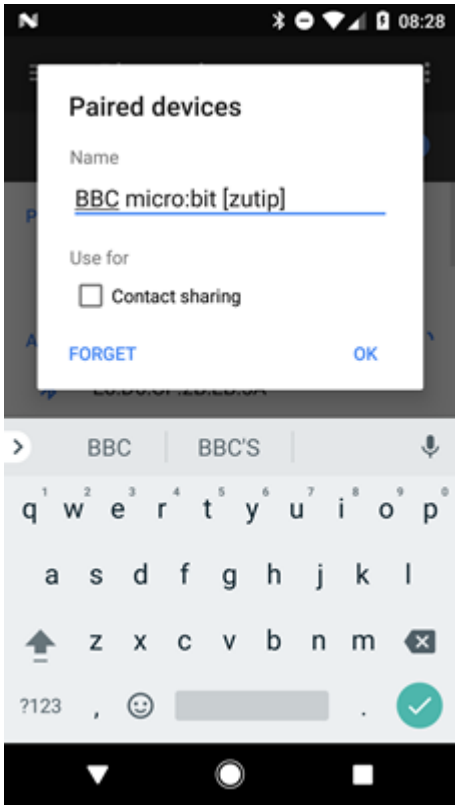


Figure 2 - Forgetting the old pairing data

2. Press and hold down both buttons A and B on the front of the micro:bit. **Keep them held down until you get to the step below where you are told to release them**. One of the most common causes of pairing problems is that people release buttons A and B too early.

3. Press and release the Reset button on the back of the micro:bit. This causes the micro:bit to reboot and as it starts up, if it finds that buttons A and B are both pressed (which they should still be!), it will enter pairing mode.

4. After about 4 or 5 seconds you should see “PAIRING MODE” scroll across the micro:bit display. **You may now release buttons A and B.**

5. When the PAIRING MODE message has completely scrolled across the micro:bit display, it will display a special pattern. This pattern is a visual identifier for your micro:bit. Now select the micro:bit on your Android smartphone’s Bluetooth settings screen under the **Available devices** heading. It will have the name BBC micro:bit [xxxxx] where “xxxxx” will be a 5 character identifier. Selecting your micro:bit on the smartphone will have one of two effects, depending on whether the hex file is using Just Works Pairing of Passkey Pairing.

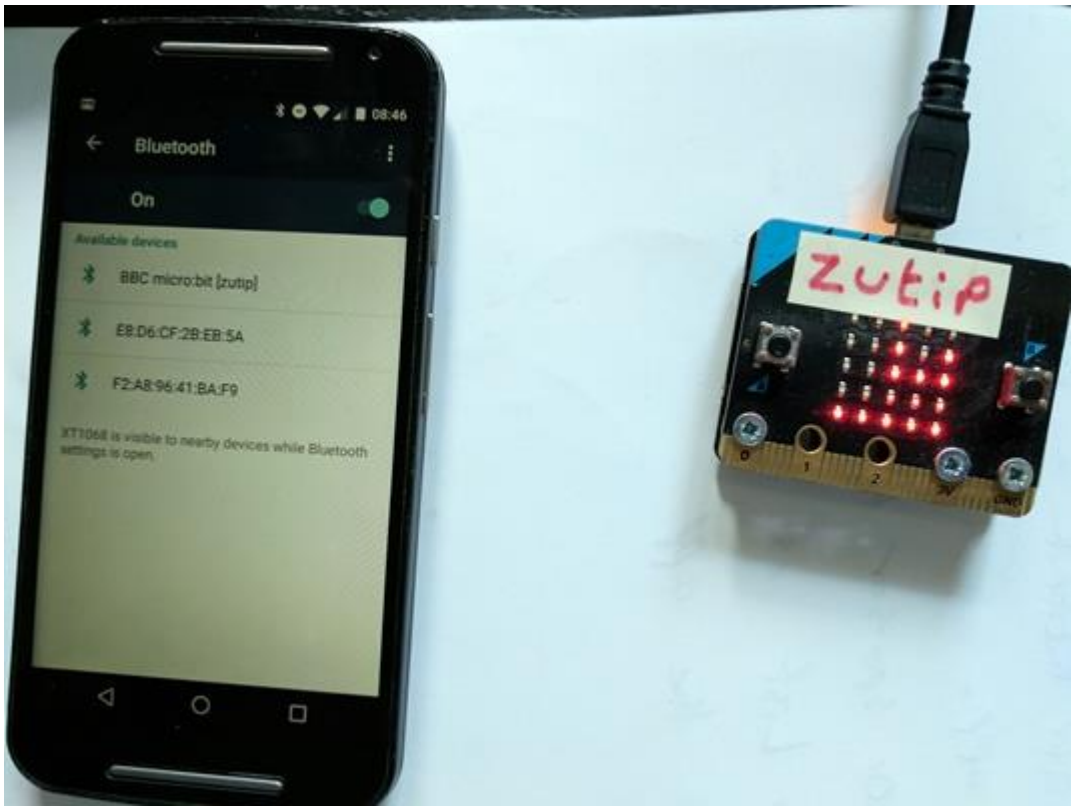


Figure 3 - ready to pair

4A. Just Works Pairing

Per the name, selecting your micro:bit on the phone, with the micro:bit displaying its “ready to pair pattern” will cause pairing to take place and complete, all in one go. If successful, a tick or check mark will appear immediately on the micro:bit.

Figure 4 - Pairing completed successfully

4B. Passkey Pairing

Passkey Pairing involves additional steps. Selecting your micro:bit on the phone, with the micro:bit displaying its “ready to pair pattern” will cause a left pointing arrow to appear on the micro:bit and an input dialogue like the one in Figure 3 to appear on the phone. The arrow is prompting you to press button A.

Figure 5 - Ready for passkey input

Pressing button A. This will cause 6 randomly selected numbers to appear one at a time on the micro:bit. You need to enter each of these digits into the smartphone input field. It may be tricky to do this at the same time as the numbers are being displayed so if necessary write them down and then enter them or have someone help you. The sequence of digits will be displayed twice and you must enter them into the phone during this time or pairing will fail. Press OK on the phone when all 6 digits have been entered. If you entered the right sequence of digits within the allowed time, a tick or check mark will appear on the micro:bit per Figure 2. If not, a cross will be displayed as shown in Figure 4.

Figure 6 - Entering the 6 digit pass key

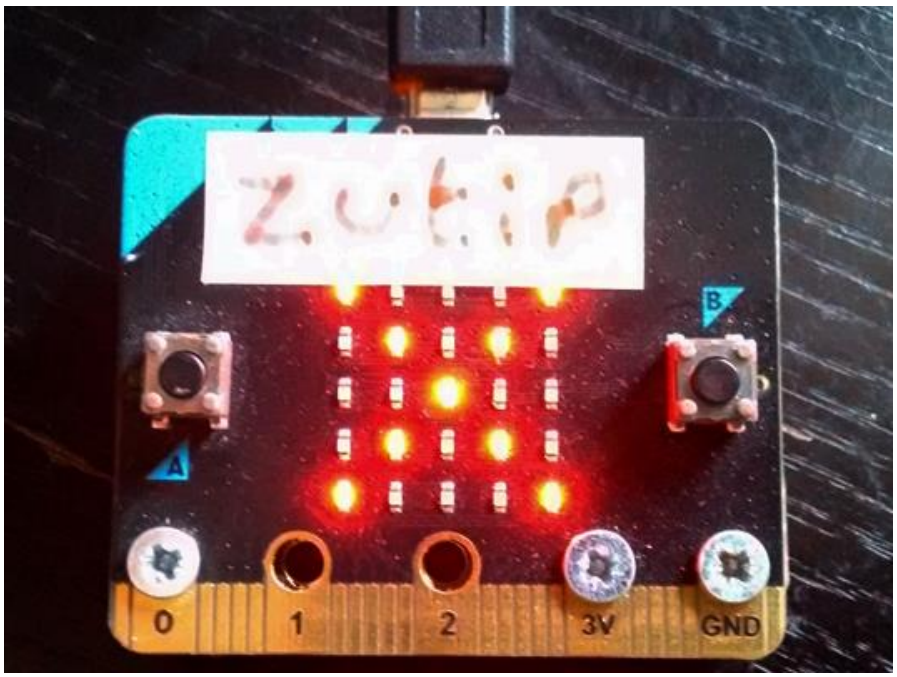


Figure 7 - Pairing failed

iOS

If you have an Android device, go to the Android section above. The procedure for Android and iOS devices is different.

1. In the Bluetooth --> Settings screen, check to see if your micro:bit is listed under My Devices. If it is, select the cog icon next to it and then select Forget. This will clear the pairing details from your phone so you can start the process afresh. A phone can be paired with more than one micro:bit at a time but if you need to repair because (say) you installed a new hex file and cleared the pairing data from the micro:bit, you *must* forget the corresponding pairing data on the phone and start again.

Figure 8 - smartphone already lists micro:bit as paired

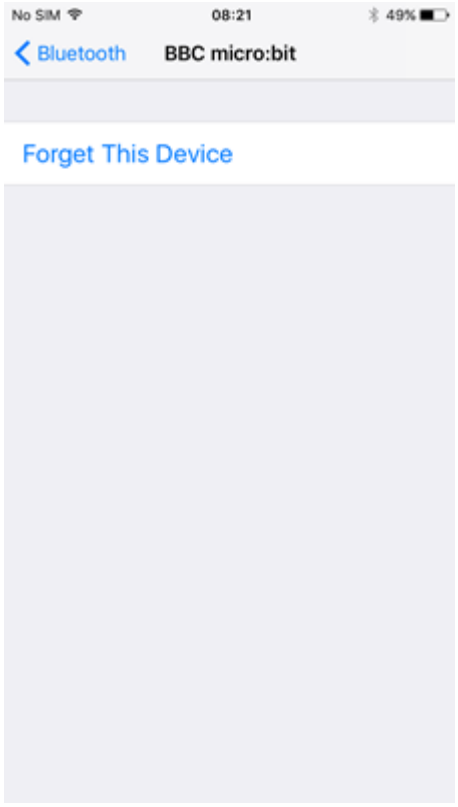


Figure 9 - so it's important to Forget the device before pairing again

Figure 10 - confirming decision to Forget the micro:bit

2. Install the nRF Connect application from the Apple App Store. We'll use this application to trigger the pairing process. Note that on an iOS device you do **not** initiate pairing from the Settings --> Bluetooth screen.

3. Press and hold down both buttons A and B on the front of the micro:bit. **Keep them held down until you get to the step below where you are told to release them**. One of the most common causes of pairing problems is that people release buttons A and B too early.

4. Press and release the Reset button on the back of the micro:bit. This causes the micro:bit to reboot and as it starts up, if it finds that buttons A and B are both pressed (which they should still be!), it will enter pairing mode.

5. After about 4 or 5 seconds you should see "PAIRING MODE" scroll across the micro:bit display. **You may now release buttons A and B.**

6. When the PAIRING MODE message has completely scrolled across the micro:bit display, it will display a special pattern. This pattern is a visual identifier for your micro:bit.

7. Launch the nRF Connect application. It will automatically scan and if your micro:bit is in pairing mode, it will be listed.

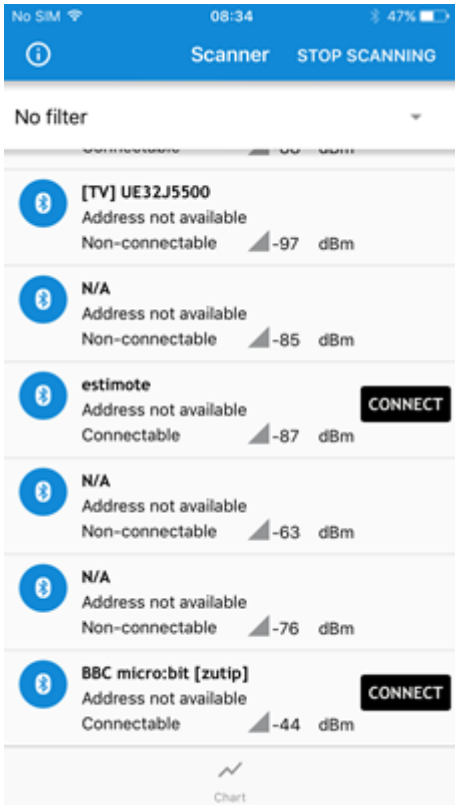


Figure 11 - micro:bit listed in the nRF Connect application

8. CONNECT to the micro:bit. The next screen will list the Bluetooth services present in the micro:bit hex file and if your micro:bit is running appropriate Bluetooth services, this will usually trigger the pairing process. Some Bluetooth services can be accessed without pairing, whilst others can only be accessed after pairing (unless the hex file stipulates that no security at all is required). If you have at least one service whose UUID starts with E95D present in your hex file, pairing should be triggered. If

you have no services of this type then you will not be able to trigger pairing, so go back to your code and ensure you include at least one of the custom micro:bit Bluetooth services whose service UUIDs start with E95D. See https://lancaster-university.github.io/microbit-docs/ble/profile/

Figure 12 - Bluetooth services

Figure 13 - Pairing initiating

9. Complete the pairing process per the Android steps in 4A or 4B above.

Raspberry Pi

See http://bluetooth-mdw.blogspot.co.uk/2017/02/pairing-bbc-microbit-with-raspberry-pi.html

7. Does your hex file allow you to enter pairing mode?

The hex file installed on a micro:bit determines whether or not pairing mode is supported.

This is usually the same as whether or not pairing is required, as covered in section 8. But if you're a C/C++ developer, you have separate settings available to you which determine whether or not pairing mode may be entered and what type, if any, of pairing is required. See section 8 for further details, noting in particular the **"pairing_mode": 0,** attribute in the config.json file.

8. Does your hex file require you to pair the micro:bit?

Whether or not you need to pair your micro:bit with your smartphone depends on the hex file installed upon it. It also determines which of the two supported pairing methods you should use when you do pair.

The programming language or tool you used to create your hex file will have its own way of letting you control the need to pair and the pairing method to be used. If you didn't create the hex file, you'll need to ask whoever provided you with the hex file, what its pairing requirements and capabilities are.

Here's a summary of the options and how they are specified for each of the more common micro:bit development tools and languages.

Microsoft MakeCode

If the Bluetooth package is not included in the MakeCode project then pairing is not required or possible.

If Bluetooth package is included in the MakeCode project then pairing settings can be found within the Project Settings page.

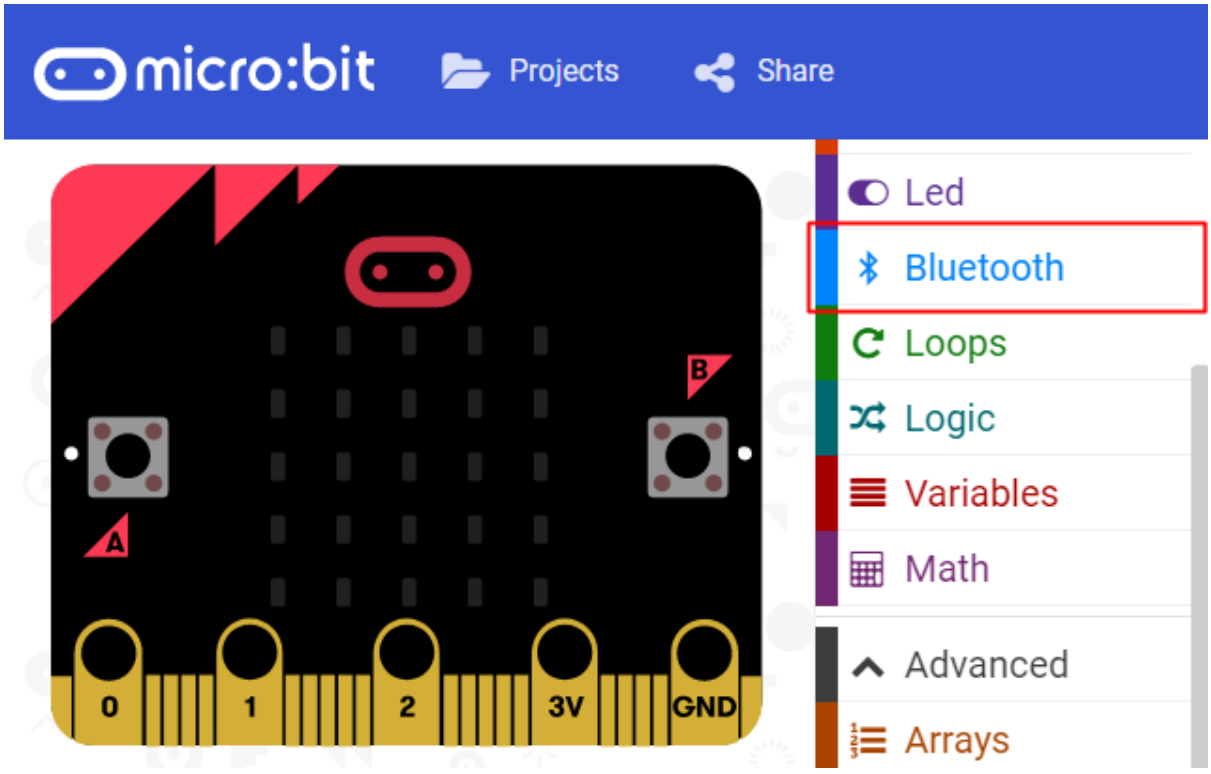


Figure 14 - MakeCode project with the Bluetooth package included

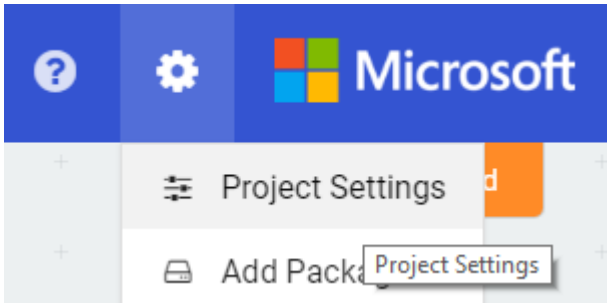


Figure 15 - Project Settings in MakeCode

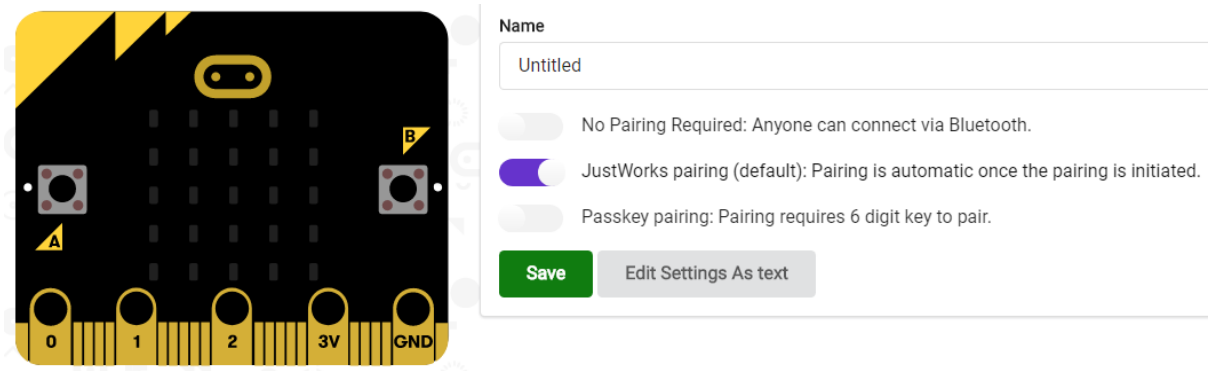


Figure 16 - Pairing options in MakeCode

The three pairing options offered in the MakeCode Project Settings screen are:

No Pairing Required : This means that pairing is not required at all and if you try to pair with your smartphone, it will not work. Specifically, any attempt to put the micro:bit into pairing mode will be ignored.

Just Works pairing : This is the default. With this option selected, pairing is achieved by putting the micro:bit into pairing mode, initiating pairing with the micro:bit from the smartphone. That's all you need to do.

Passkey pairing: This mode will require you to press a button on the micro:bit after pairing has been initiated on the smartphone. A 6 digit PIN will be displayed, one digit at a time, and each digit must be entered into a pop-up dialogue which should appear on the smartphone.

C/C++ using Yotta

The project file config.json controls whether or not it is possible to enter pairing mode, whether or not pairing is required and if it is, whether passkey (PIN-based) pairing or "just works" pairing is required. Consider the following example:

```
{
    "microbit-dal": {
        "bluetooth": {
            "enabled": 1,
            "pairing_mode": 0,
            "private_addressing": 0,
            "open": 0,
            "security_level": "SECURITY_MODE_ENCRYPTION_WITH_MITM",
            "whitelist": 1,
            "advertising_timeout": 0,
            "tx_power": 4,
```

```
            "dfu_service": 0,
            "event_service": 1,
            "device_info_service": 1
        },
        "gatt_table_size": "0x700"
    }
}
```

microbit-dal.bluetooth.enabled : 1 means Bluetooth is enabled, 0 means it is disabled

microbit-dal.bluetooth.pairing_mode : 1 means it will be possible to put the micro:bit into pairing mode. 0 means it will not.

microbit-dal.bluetooth.open : 1 means no pairing is required. 0 means pairing of a type indicated by

microbit-dal.bluetooth.security_level : SECURITY_MODE_ENCRYPTION_NO_MITM Just Works pairing will be used when pairing. SECURITY_MODE_ENCRYPTION_WITH_MITM means passkey (PIN-based) pairing will be used.

C/C++ using the mbed web IDE

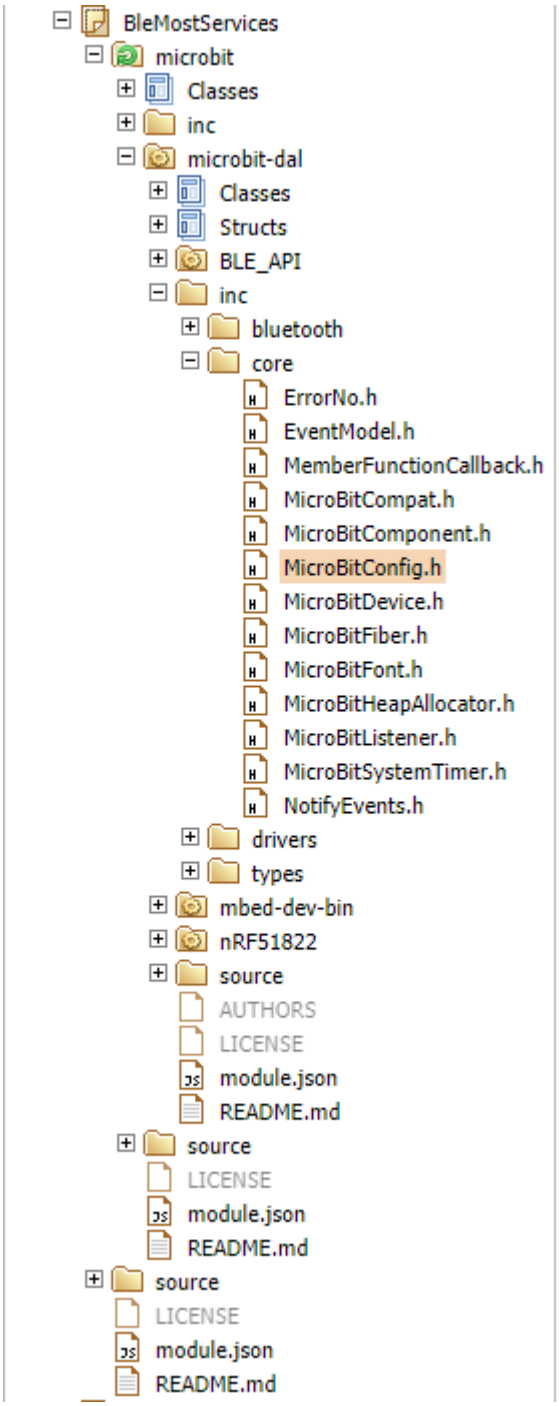


Figure 17 - MicroBitConfig.h in an mbed IDE project

The mbed IDE does not use the config.json file which is used with the offline Yotta build system described in the previous section. Bluetooth pairing and other options are instead configured using constants which are in the microbit/microbit-dal inc/core/MicroBitConfig.h source file.

All Bluetooth related constants can be found under a comment "// BLE options". The following constants are relevant to the subject of pairing:

```
// Enable/Disable BLE during normal operation.
// Set '1' to enable.
#ifndef MICROBIT_BLE_ENABLED
#define MICROBIT_BLE_ENABLED                    1
#endif
```

MICROBIT_BLE_ENABLED: Purpose and use per comment in code.

```
// Enable/Disable BLE pairing mode mode at power up.
// Set '1' to enable.
#ifndef MICROBIT_BLE_PAIRING_MODE
#define MICROBIT_BLE_PAIRING_MODE               1
#endif
```

MICROBIT_BLE_PAIRING_MODE: the comment in the code is a little unclear regarding "at power up". This constant controls whether *it is possible to enter pairing mode at power up* by holding down buttons A+B. 1 means it will be possible to put the micro:bit into pairing mode. 0 means it will not.

```
// Convenience option to enable / disable BLE security entirely
// Open BLE links are not secure, but commonly used during the
development of BLE services
// Set '1' to disable all secuity
#ifndef MICROBIT_BLE_OPEN
#define MICROBIT_BLE_OPEN                       1
#endif
```

MICROBIT_BLE_OPEN: 1 means no pairing is required. 0 means pairing of a type indicated by the constant MICROBIT_BLE_SECURITY_LEVEL will be required.

```
// Define the default, global BLE security requirements for MicroBit BLE
services
// May be one of the following options (see mbed's SecurityManager class
implementaiton detail)
// SECURITY_MODE_ENCRYPTION_OPEN_LINK:      No bonding, encryption, or
whitelisting required.
// SECURITY_MODE_ENCRYPTION_NO_MITM:        Bonding, encyption and
whitelisting but no passkey.
// SECURITY_MODE_ENCRYPTION_WITH_MITM:      Bonding, encrytion and
whitelisting with passkey authentication.
//
#ifndef MICROBIT_BLE_SECURITY_LEVEL
#define MICROBIT_BLE_SECURITY_LEVEL
SECURITY_MODE_ENCRYPTION_WITH_MITM
#endif
```

MICROBIT_BLE_SECURITY_LEVEL: SECURITY_MODE_ENCRYPTION_NO_MITM Just Works pairing will be used when pairing. SECURITY_MODE_ENCRYPTION_WITH_MITM means passkey (PIN-based) pairing will be used. SECURITY_MODE_ENCRYPTION_OPEN_LINK means no pairing is required. Note that it is more typical to set MICROBIT_BLE_OPEN to 1 if you do not want pairing to be required.

Microsoft Blocks, Touch Develop and the CodeKingdoms editors at the BBC web site

The BBC originally launched micro:bit with a selection of code editors. Touch Develop, Blocks and the CodeKingdoms editors produced hex files with Bluetooth enabled, passkey pairing required and only certain Bluetooth services included. From memory, these were the Device Information Service, the Event Service and the Device Firmware Update (DFU) Service. Only specific "canned" Bluetooth use cases were supported, such as triggering a connected smartphone to take a photograph. For Bluetooth use, therefore, these editors are not recommended. Microsoft MakeCode and C/C++ using Yotta or the mbed IDE are the recommended editors for developers wishing to use Bluetooth.

Furthermore, the BBC editors have recently been deprecated and no longer maintained.

Our editors have had some exciting updates. The JavaScript Blocks Editor is now at makecode.microbit.org, so make sure that URL isn't blocked in your school.

You can still access the legacy code editors and migrate your programs, but they are no longer being updated.

Developers are being encouraged to migrate to MakeCode.

https://support.microbit.org/support/solutions/articles/19000050402-can-i-still-access-touchdevelop-microsoft-blocks-and-codekingdoms-and-old-python-editors-/

In general, if a developer is having Bluetooth related problems with code which was developed one of these legacy editors, the first response should be to instruct them to recreate using MakeCode or C/C++, test again and then resume troubleshooting if the problem persists.

micropython

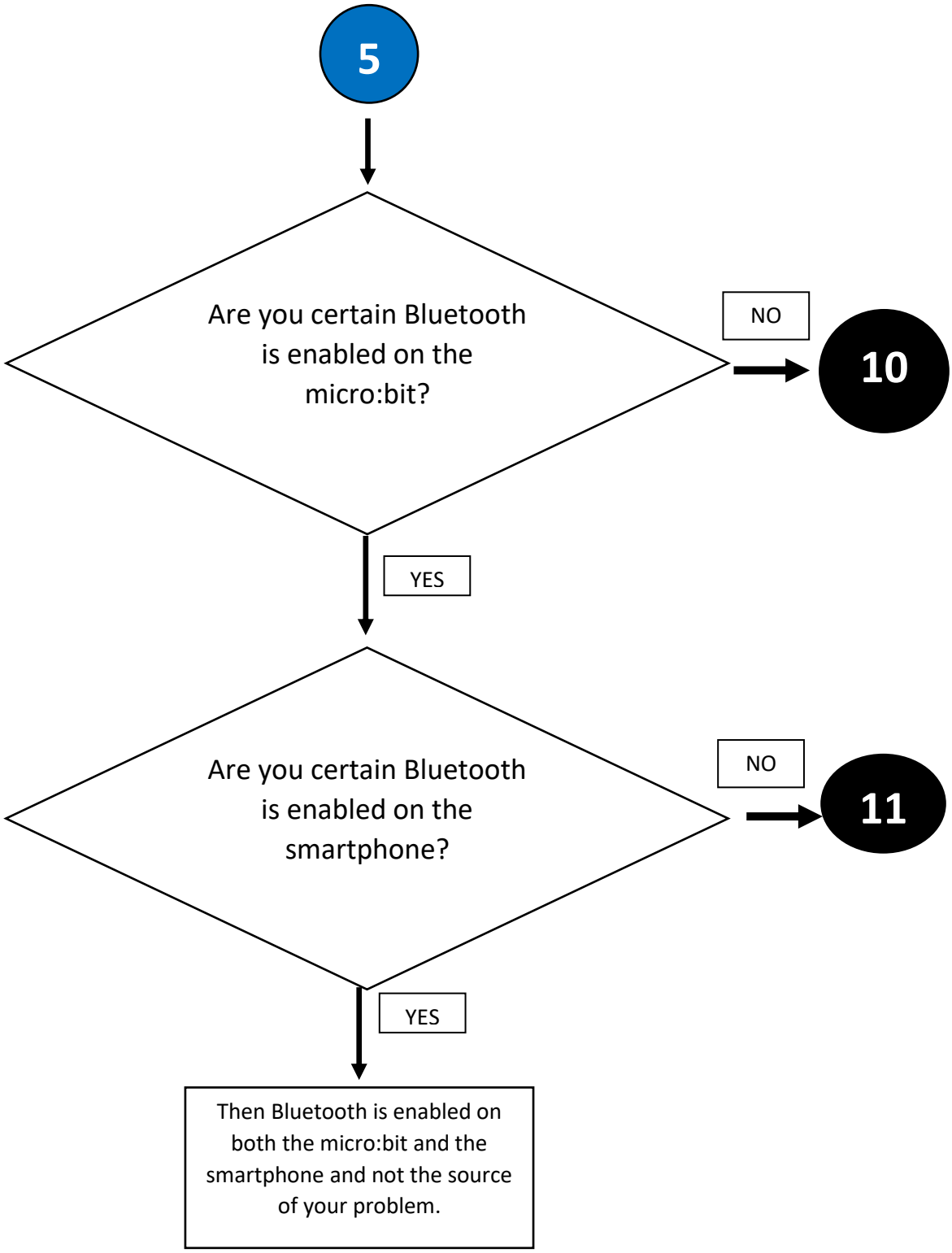
Bluetooth is not supported and so no pairing is required or possible.

9. Is Bluetooth switched on?

9

BLUETOOTH ON - micro:bit

BLUETOOTH ON - smartphone



10. Is Bluetooth enabled on the micro:bit?

The BBC micro:bit supports Bluetooth but it is only available for use if the right kind of firmware (hex file) is installed on the micro:bit and the programming language or tool you used to create the hex file supports Bluetooth. The table below summarises how to enable Bluetooth when using the various tools for hex file creation.

Tool / Language	Bluetooth Supported?	How is Bluetooth enabled?
Microsoft MakeCode	YES	Add the Bluetooth package to your project and remove the "radio" package.
C/C++ using Yotta	YES	microbit-dal.bluetooth.enabled=1 in config.json
C/C++ using the mbed web IDE	YES	set MICROBIT_BLE_ENABLED to 1 in MicroBitConfig.h
Blocks at the BBC web site	YES but with significant restrictions	Enabled by default
TD at the BBC web site	YES but with significant restrictions	Enabled by default
micropython	NO	Not supported

Note that Bluetooth is supported by the BBC tools but only for very limited, predefined applications. MakeCode is the recommended graphical programming tool for Bluetooth applications. C/C++ is the recommended textual programming language.

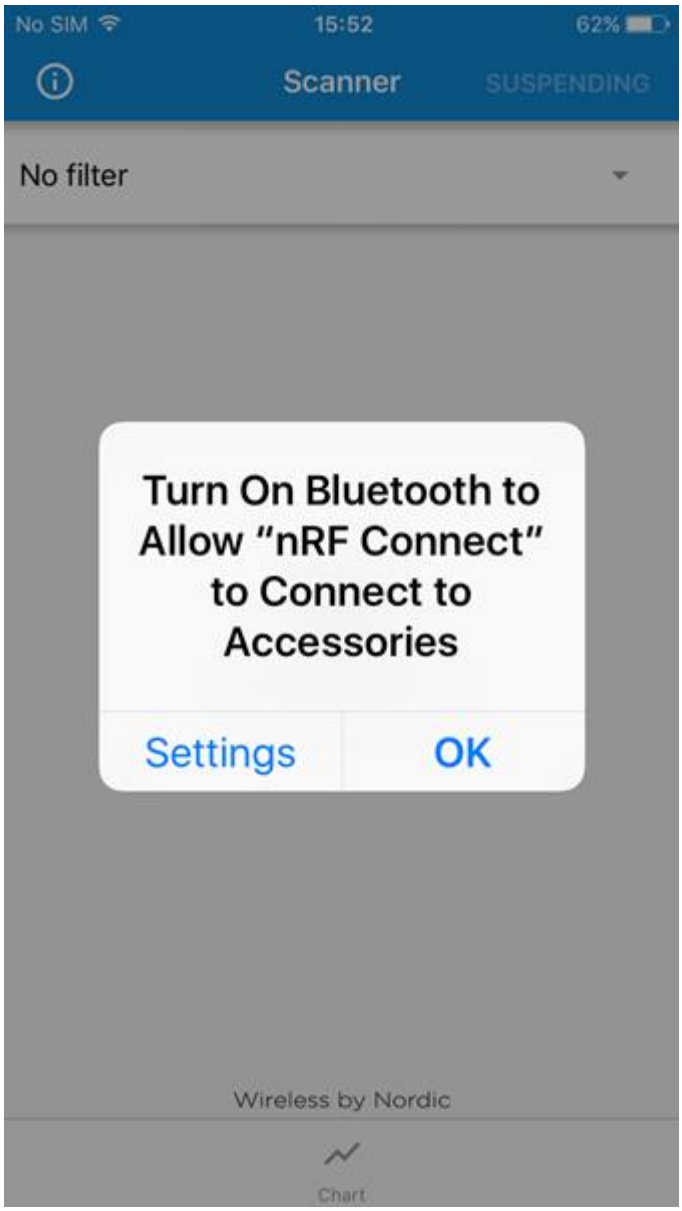
11. Is Bluetooth enabled on the smartphone?

All smartphones and tablets allow Bluetooth to be switched off. It can either be directly switched on or off in the Settings / Bluetooth screen of the smartphone or as a consequence of enabling or disabling “flight mode”.

Some smartphones disable Bluetooth by default immediately after switching on. Some enable it by default.

Usually an icon on the smartphone will indicate whether or not Bluetooth is enabled.

If you’re not sure, try installing the nRF Connect application from Google Play or the Apple App store and running it. If Bluetooth is switched off, it will inform you.



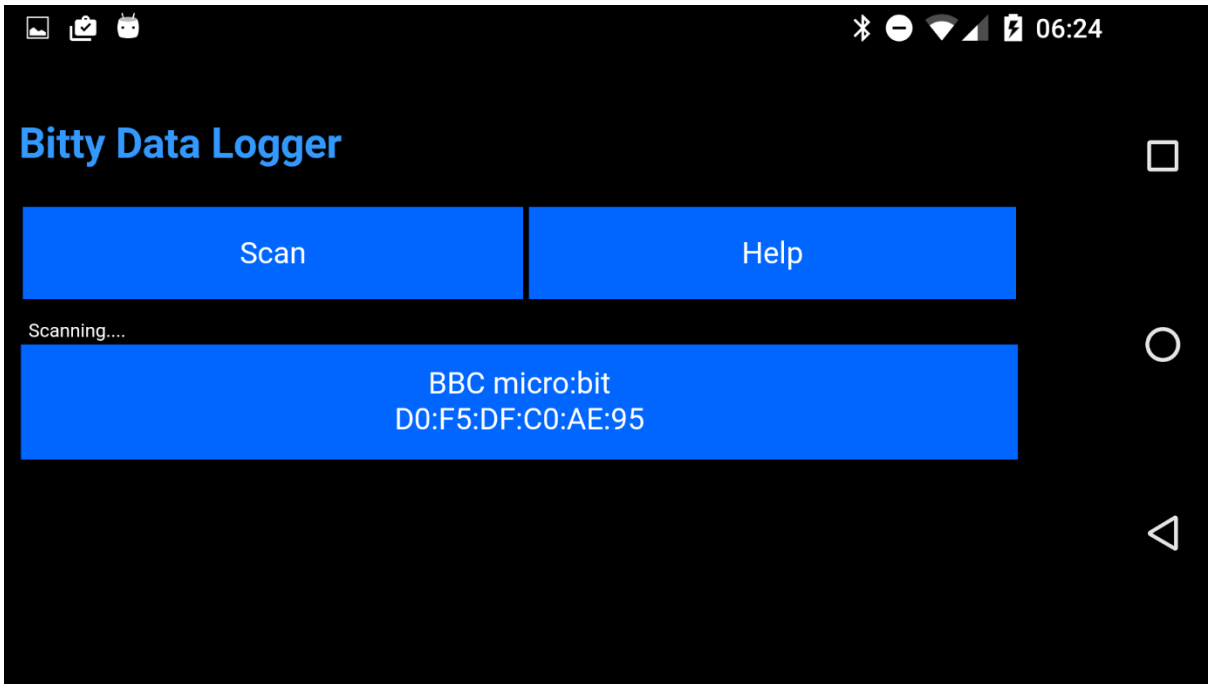
12. Problems finding a micro:bit (scanning)

The typical sequence of operations when using a smartphone to interact with a micro:bit over Bluetooth is as follows:

(1) Scan ---> (2) Connect ---> (3) Use

(1) Scan

Scanning involves the smartphone receiving Bluetooth transmissions called advertisements, from nearby Bluetooth devices. A micro:bit needs to advertise its availability before it can be connected to and used. Provided the hex file installed on it has Bluetooth enabled and if required, you've paired (see 7, 8 and 10) then it should advertise immediately after a reset and keep advertising until it is connected to. Smartphone applications may look for the text "BBC" in advertising data to select those devices which seem to be micro:bits and to list those devices on the smartphone screen where you would usually be required to make a selection.



If scanning does not result in your micro:bit being listed, consider the following:

1. Does your micro:bit require you to pair? If yes then have you paired? YES – your micro:bit should be advertising. NO – will not be advertising so pair with it first before scanning.

2. Try scanning for your micro:bit with another application. "nRF Connect" is a free application and is available for both Android and iOS devices. Install it from Google Play or the Apple App Store and run it. It will start scanning immediately if Bluetooth is enabled. If it finds your micro:bit then the problem is with your other smartphone application. If it does not, then your micro:bit is not advertising. This would be caused either by the hex file requiring you to pair and you not having paired or the hex file not having Bluetooth enabled.

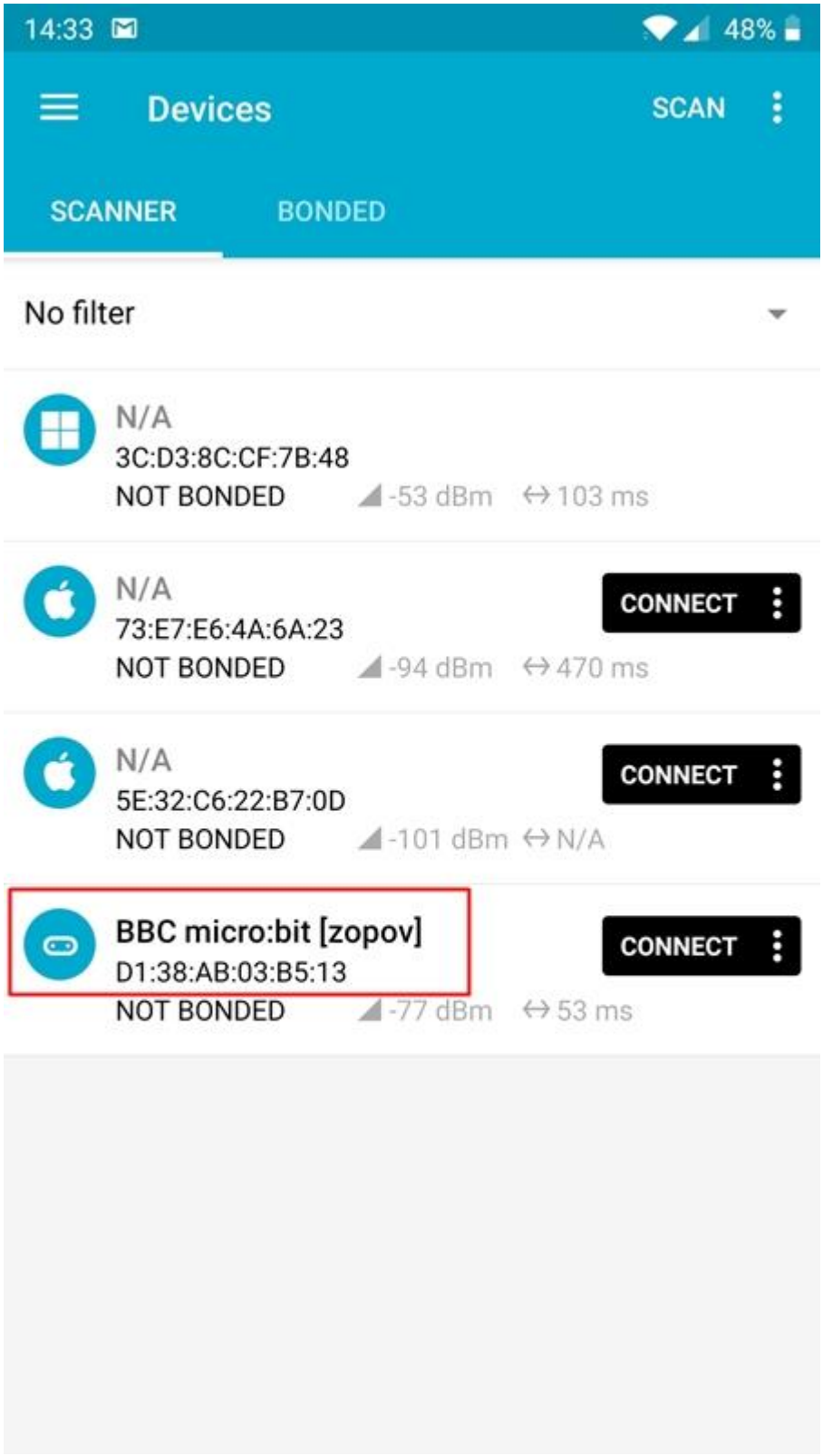
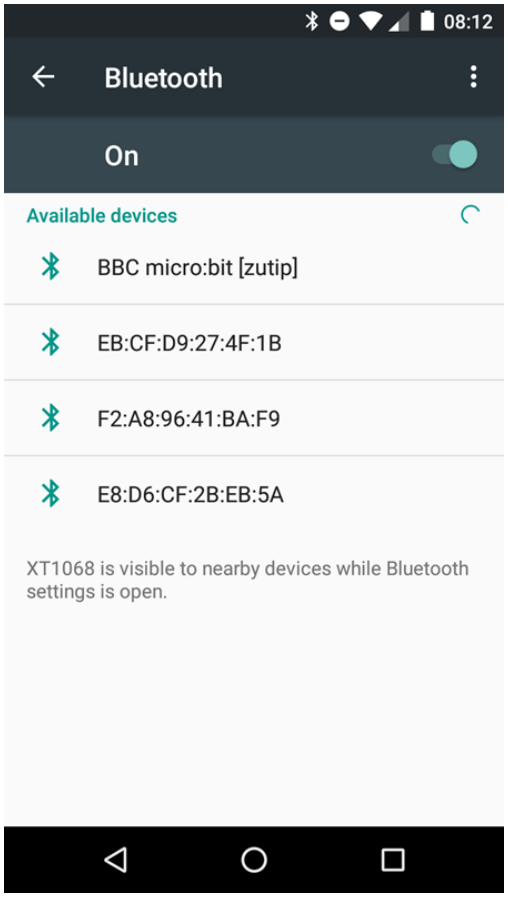


Figure 18 - A micro:bit discovered by nRF Connect, proving the micro:bit is advertising

3. On an Android device, go into Settings --> Bluetooth. Switch Bluetooth off. Switch it on again. It should now be scanning and list Bluetooth devices nearby which are scanning. If your micro:bit is listed then Bluetooth is working properly on both the smartphone and micro:bit, which suggests your problem is with your application. If no Bluetooth devices are listed, this suggests a problems with your phone or the application itself.



4. Android phones require you to give applications which perform Bluetooth scanning, the "Location" permission. This may seem a little weird and you may be concerned that it means that the application is going to track your location. If you're concerned about this, you will need to ask the application developer what the application does with the location permission. The Bitty Software applications require this permission solely so that Bluetooth scanning to find micro:bits can be performed. Your location is not tracked in any way.

5. Reset your micro:bit. If something else has connected to your micro:bit then that will stop it advertising and hence prevent it from being discovered when scanning. Reseting will disconnect the other device.

6. Some older phones do have occasional reliability problems with Bluetooth. Attempt to resolve the issue by switching your phone off, switching it on again and then going into Settings --> Bluetooth and making sure Bluetooth is switched on. Make repeated attempts. It may be that your old phone is a little slow and repeated scanning attempts are required.

13. Problems connecting to a micro:bit

The typical sequence of operations when using a smartphone to interact with a micro:bit over Bluetooth is as follows:

(1) Scan ---> (2) Connect ---> (3) Use

(2) Connect

Selecting a micro:bit which has been discovered through scanning and listed on a smartphone application screen, will typically result in the smartphone trying to connect to the micro:bit. Connecting is usually, but not always, required for a smartphone to interact with a micro:bit. Connecting can fail under some circumstances.

A possible cause of a smartphone being unable to connect to a micro:bit is that another device is already connected to it. micro:bit can accommodate a connection from no more than one other device at a time. If you scan from two different smartphones, each will discover and display the micro:bit. Selecting the micro:bit one one of the smartphones should result in it connecting. Trying to connect from the other smartphone should fail.

To forcibly disconnect another device from your micro:bit, press the reset button.

Tip: if you are developing your own hex files, use the Bluetooth connection event handlers to display a “C” when a connection is accepted and a “D” when disconnected. This will allow you to identify what state your micro:bit is in with respect to Bluetooth connections.

Table 1 - C/C++ showing use of Bluetooth connection events

```
void onConnected(MicroBitEvent)
{
    uBit.display.print("C");
}

void onDisconnected(MicroBitEvent)
{
    uBit.display.print("D");
}

int main()
{
    // Initialise the micro:bit runtime.
    uBit.init();

    uBit.display.scroll("BLUE-JW");

    uBit.messageBus.listen(MICROBIT_ID_BLE, MICROBIT_BLE_EVT_CONNECTED,
onConnected);
    uBit.messageBus.listen(MICROBIT_ID_BLE, MICROBIT_BLE_EVT_DISCONNECTED,
onDisconnected);
```

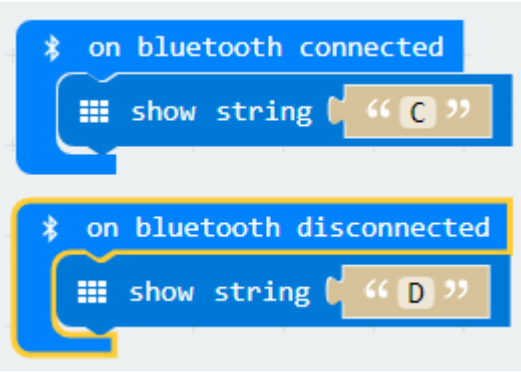


Figure 19 - MakeCode and Bluetooth connection event handlers

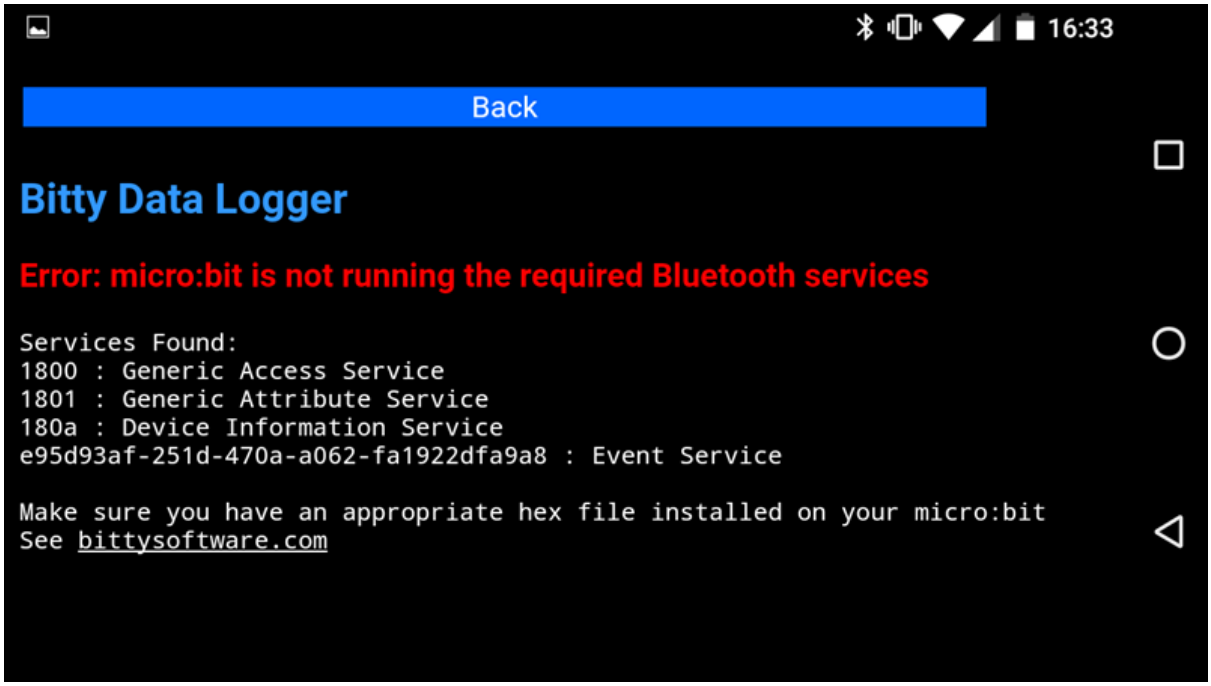
Note that it's possible your smartphone *is* connecting to the micro:bit but is then disconnected shortly afterwards. This case is covered on page 15.

14. Missing services

The typical sequence of operations when using a smartphone to interact with a micro:bit over Bluetooth is as follows:

(1) Scan ---> (2) Connect ---> (3) Use

(3) Use



Bluetooth Low Energy includes a concept known as a “service”. A service is a module which provides a particular set of functionality over Bluetooth. Several services have been defined for micro:bit and information is available in the micro:bit documentation, here: https://lancaster-university.github.io/microbit-docs/ble/profile/

Services are software modules and they are only present in your micro:bit hex file if the hex file developer included them. A few services are included by default, depending on the development tool being used, but in most cases they must be explicitly included. How you do this will vary according to the development tool.

```
int main()
{
    // Initialise the micro:bit runtime.
    uBit.init();

    uBit.display.scroll("BLUE-JW");

    uBit.messageBus.listen(MICROBIT_ID_BLE, MICROBIT_BLE_EVT_CONNECTED,
onConnected);
    uBit.messageBus.listen(MICROBIT_ID_BLE, MICROBIT_BLE_EVT_DISCONNECTED,
onDisconnected);

    new MicroBitAccelerometerService(*uBit.ble, uBit.accelerometer);
```

```
    new MicroBitButtonService(*uBit.ble);
    new MicroBitIOPinService(*uBit.ble, uBit.io);
    new MicroBitLEDService(*uBit.ble, uBit.display);
    new MicroBitMagnetometerService(*uBit.ble, uBit.compass);
```

Figure 20 - Example of Bluetooth services being included in C/C++

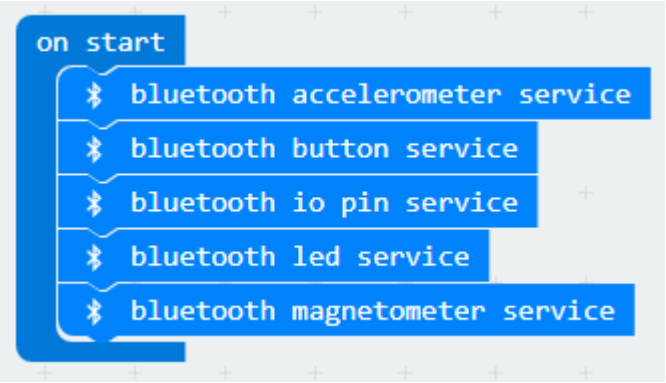


Figure 21 - Services being included in a MakeCode project

If an application indicates that there are required services missing from the micro:bit it is connected to, there are two possibilities:

14A. Services Are Missing

The first possibility is that the hex file genuinely does not contain the services required by the smartphone application. If you're the hex file developer, you need to fix this in your code. If you're not, you need to obtain a replacement hex file from the developer.

14B. Service Cache is Out of Date

The second possibility is that the micro:bit has the right services but the smartphone is operating from a local cache of services, which does not reflect the true state of the micro:bit. To solve this, the cache must be refreshed. Forgetting pairing details and then switching Bluetooth off and then on again and then scanning for and connecting to the micro:bit using an application, may work. nRF Connect for Android has a Refresh Services option in the menu which is available after you connect to the micro:bit and this usually works.

15. Immediate disconnection

If your smartphone application seems to successfully connect but then very shortly afterwards is disconnected, it's possible that the smartphone thinks it is paired to the micro:bit but the micro:bit does not. In this case, the smartphone will attempt to establish an encrypted connection with the micro:bit, but because the micro:bit is not in a paired state, it lacks the required encryption keys and the process will fail, causing the micro:bit to drop the connection.

Every time you flash a new hex file over USB to your micro:bit, you will lose pairing information and will need to repair. So if you are using pairing, every time you flash new code over USB to the micro:bit, "Forget" the pairing on your smartphone and go through the pairing procedure again so that both smartphone and micro:bit are paired with each other and both possess the associated pairing data.

Note that flashing over Bluetooth from applications such as the "official" micro:bit smartphone applications for Android and iOS will not clear the pairing keys and so will not require pairing to be performed again.

16. GATT Errors

Sometimes smartphone applications will report GATT errors such as “GATT Error 133”. This is a catch-all error message which means that the micro:bit closed the Bluetooth connection for some reason. Unfortunately no more information than this tends to be available. If this happens, reflash the micro:bit, Forget pairing details on the smartphone and restart it and, assuming you are using pairing, pair again before re-testing.

17. micro:bit to micro:bit

It is possible to use Bluetooth to communicate directly with another micro:bit but this is difficult to set up and leaves little memory left for other purposes. Note that the “radio” API, which gives access to a simpler, non-standard wireless capability that the micro:bit has, is a much easier way to accomplish direct, wireless communication between micro:bits and unless you have a particular reason for wanting to use Bluetooth specifically, you are recommended to use Radio instead in this situation.

To understand the issues and opportunities regarding Bluetooth and micro:bit to micro:bit communication, read about Bluetooth device roles here:

http://bluetooth-mdw.blogspot.co.uk/2016/07/microbit-and-bluetooth-roles.html

By default, a micro:bit is a Bluetooth GAP Peripheral. It can advertise and accept connections from a Bluetooth GAP Central device but it cannot scan and form connections itself. For a micro:bit to be able to connect to another micro:bit, one must act as GAP Peripheral and the other as GAP Central.

The micro:bit has Bluetooth stack from Nordic Semiconductor. They have several binaries available, called Soft Devices. S110 allows a device to act as a GAP Peripheral. S120 allows a device to act as a GAP Central. S130 allows the device to act in either / both of these roles.

All available development tools use S110 by default. If you want to use S120 or S130, your only option is to roll up your sleeves and start working in C/C++ with the microbit-dal code. Details on how to accomplish this are beyond the scope of this guide.

18. Too many devices paired

You can pair a maximum of 4 devices such as smartphones and tablets with one micro:bit. If you try to pair a fifth device, without having first cleared the pairing data for the other four devices by flashing a hex file to the micro:bit, pairing will fail. The symptoms of the failure will vary from smartphone to smartphone but may involve an error such as GATT Error 133.

In this situation your choices are to accept that you cannot pair a fifth device or to clear details of the original four devices from the micro:bit by flashing the hex file over USB and then pairing your new smartphone, which when successfully paired, will make it the only device paired with this micro:bit. You should “Forget” the pairing details off the other four devices to avoid confusion at this point.

Note that this procedure relies on flashing over USB and not Bluetooth, since flashing over USB will clear previous pairing details from the micro:bit whereas flashing over Bluetooth will not.