Chapter 5A: Classic Bluetooth – The Wireless Serial Port

Time: 3 ¾ Hours

At the end of this chapter you will understand the basics of Classic Bluetooth and how to create a simple Classic Bluetooth project on WICED devices. This section is focused on the simplest Bluetooth connection, one Master (Android, Mac or PC) and one Slave (your WICED Bluetooth Device). By the end you should understand Inquiry, Page, Pair, Bond, SDP, L2CAP, RFCOMM and the Serial Port Profile (SPP).

9.1 WICED Bluetooth Classic System Lifecycle 2

9.1.1 Inquiry 3

9.1.2 Page / Connect 4

9.1.3 Pair & Bond 4

9.1.4 Discover the Services using Service Discovery Protocol (SDP) 4

9.1.5 Exchange Data with the Serial Port Profile 5

9.2 Secure Simple Pairing 5

9.3 Service Discovery Protocol (SDP) 5

9.4 L2CAP, RFCOMM & the Serial Port Profile 6

9.4.1 L2CAP 6

9.4.2 RFCOMM 7

9.4.3 Serial Port Profile 7

9.5 WICED Bluetooth Designer 7

9.5.1 WICED Studio 6.2 – BT Designer Bugs 16

9.6 WICED Bluetooth Stack Events 16

9.7 WICED Classic Bluetooth Firmware Architecture 17

9.7.1 Initialization Functions 18

9.7.2 SDP Database 18

9.7.3 Handle Pairing 20

9.7.4 Handle Bonding 21

9.7.5 Serial Port Profile 23

9.8 Exercises 26

Exercise - 5A.1 Create a Serial Port Profile Project 26

Exercise - 5A.2 Improve Security by Adding IO Capabilities 35

Exercise - 5A.3 Add Multiple Device Bonding Capability 35

# WICED Bluetooth Classic System Lifecycle

The Bluetooth Classic Spec has a bewildering amount of complexity. Clearly this must have been one of the motivations for creating the much simpler BLE standard. Like Chapter 4 we will take the approach of creating the simplest example project possible to get things going.

The simplest Bluetooth Classic scenario has two devices, a Master and a Slave. Slaves are passive – not transmitting – until they hear an Inquiry broadcast from a Master, at which point the Slave broadcasts basic information about itself (Name, BDADDR, Services). The Master then Pages (connects) to the Slave and they exchange and save Pairing information. The Master then discovers the Services i.e. capabilities of the Slave. Finally, a basic wireless Serial Port data exchange connection is created.

The five steps are:

1. Inquiry – Master finds a Slave to Connect
2. Paging – Master connects to Slave
3. Pair & Bond – A secure, authenticated connection is created
4. Service Discovery (SDP) – The Master figures out what the Slave can do
5. Exchange Data using the Serial Port Profile

The architecture of a Bluetooth Classic device is essentially the same as that of a BLE device. It is composed of the same four layers.

|  |  |  |
| --- | --- | --- |
|  | Application | The code that you write to implement your system functionality |
| Bluetooth Classic Stack | Host | Provides multiple connection paths with to the application each with its own properties (reliable, ordered, time critical etc). It also provides Services to the local and remote application |
| Controller | Establishes and maintain links between devices |
| Hardware | Radio | RF magic & the best reason to use Cypress chips |

Here is the overall picture of the simplest Bluetooth Classic system.



## Inquiry

The purpose of the Inquiry process is for a Bluetooth Master to find all the Bluetooth Slaves that are within its radio range that might provide some interesting Service. This is exactly the opposite of BLE where a Peripheral advertises it availability and the BLE Central Scans for those packets.

A Bluetooth Classic Slave sits in state called Inquiry Scan i.e. a listening only state, until it hears a Bluetooth Master broadcast an inquiry request message. The Slave application is responsible for putting the Stack into the Inquiry Scan state using the correct Stack API.

Upon hearing an Inquiry request the Slave will broadcast an Extended Inquiry Response (EIR) packet that contains its Name, Bluetooth Address (BDADR) and list of Services). These responses are handled completely by the Controller part of the Stack i.e. your Application is not aware of these Inquiry requests happening.

You should be aware that because of the vagaries of the Bluetooth Radio frequency hopping scheme, these Inquires make take up to 10ish seconds.

## Page / Connect

The Paging process is used for a Bluetooth Master to connect to a Bluetooth Slave. The Master is the "paging” the Slave device (remember the old school [pagers](https://www.youtube.com/watch?v=l7Og1DuMu3k&list=RDl7Og1DuMu3k&t=18)?).

A Bluetooth Classic Slave sits in state called Page Scan i.e. a listening only state, until a Bluetooth Master initiates the connection process by sending a Page request. The Slave application is responsible for putting the Stack into the Page Scan state using the correct Stack API.

A Slave can be in both the Page Scan and Inquiry Scan mode at the same time. Meaning a Master can initiate a connection to a Slave without Inquiring if it already knows of the existence of the Slave from a previous connection.

## Pair & Bond

The whole Bluetooth communication system depends on having a shared symmetric encryption key called the Link Key. Bluetooth Classic uses a process called Secure Simple Pairing that exchanges enough information the Link Key to be created.

The Secure Simple Pairing process was designed to minimize the chances that the communication link could be compromised by an eavesdropper or by a man-in-the-middle. The process is the same as the BLE process minus the Numeric Comparison method.

As with BLE, Bonding is just saving the BDADR/Link Key into a non-volatile memory so that it can be reused to speed up the re-initiating of a connection

## Discover the Services using Service Discovery Protocol (SDP)

A simple conceptual model of Bluetooth Classic device is a Server that is running one or more Services that are attached to Ports, the same model that we use in IP Networking.

One question that arises from this idea is how do I figure out “What Services are available and what Port are they listening on?”. The answer to both questions is the Service Discovery Protocol.

The SDP has a database embedded in it that contains a list of Service and what Port they are running on. And the SDP Protocol allows the Bluetooth Master to query the SDP database.

## Exchange Data with the Serial Port Profile

Once the Service Discovery is complete, the Bluetooth Master knows the Port number that it can connect to use the Serial Port Profile (SPP). The SPP is just one of these Servers (from the last section) that acts like a serial port. You put bytes in one side and they come out the other.

The Bluetooth Master then opens a connection to the SPP Server running on the Bluetooth Slave. At which point you can commence the final step in your first basic project, actually exchanging data.

# Secure Simple Pairing

Secure Simple Pairing is the same Pairing technique that we used in the BLE. You use a PIN code which is

1. Either ‘0000’ or trivial if you have no I/O capability
2. Displayed on one side, then entered on the other
3. Transmitted out of band (e.g. NFC)

The PIN is then used to encrypt random numbers which are generated on both side of the connection.

Finally, the Pin + Random Numbers + some data about the device are combined into a Link Key which serves as a shared secret to identify and encrypt data between the devices.

# Service Discovery Protocol (SDP)

From the Bluetooth Core Spec – “The service discovery protocol (SDP) provides a means for Applications to discover which Services are available and to determine the characteristics of those available services.” The SDP sits on top of the L2CAP layer – and when communicating generates a bunch of L2CAP traffic.

The Bluetooth SIG specifies that SDP database format in Volume 3 Part B of the Bluetooth Core Spec. The database is composed of one or more Service Records each containing one or more Service Attributes. Each Service Attribute is a Key/Value pair. There are several Bluetooth Sig Specified Service Attributes, in addition you can create custom Attributes.

Some of the legal Attributes include:

ServiceRecordHandle – A 32-bit number uniquely identifying that Service in the SDP

ServiceClassIDList – Identifies what type of Service this record represents, specifically a list of classes of Service

ProtocolDescriptorList – A list of the protocol stacks that may be used to access this service

ServiceName – A plain text description of the Service

The protocol provides for the Client to Search for Services and Attributes, and request the values of the same.

# L2CAP, RFCOMM & the Serial Port Profile

The Bluetooth Classic system has a stack of software and hardware built into it. For the purposes of this simple Bluetooth Classic example, three blocks in the Host are relevant, L2CAP, RFCOMM and the Serial Port Profile.

You can see the three blocks in this simplified diagram of the Stack.



## L2CAP

L2CAP is an acronym that stands for Logical Link Control and Adaptation-layer Protocol. L2CAP has one main function in the system. Specifically, it serves as a data packet multiplexor that lets you have streamed multiple connections from the higher level going into one interlaced set of packets going out the Radio. It obviously implements the de-multiplexor function as well, taking a single stream of packets and turning it back into complete streams on the other side of the link.

The L2CAP divides up the streams of data into L2CAP Channels that:

1. Divides up streams of data into smaller packets that will fit through the Radio
2. Provides quality of service to each of the L2CAP channels
3. Provide flow control

## RFCOMM

RFCOMM was built as a wired RS232 replacement protocol. The protocol support all the normal wires for a serial port include Rx, Tx, CTS, RTS, DSR, DTR, CD and Ri. Depending on the implementation, RFCOMM gives you up to 60 Server Channels of streams of serial data. The protocol is built on top of the L2CAP (a packet based system). It appears to the Application developer with APIs that look like a UART.

## Serial Port Profile

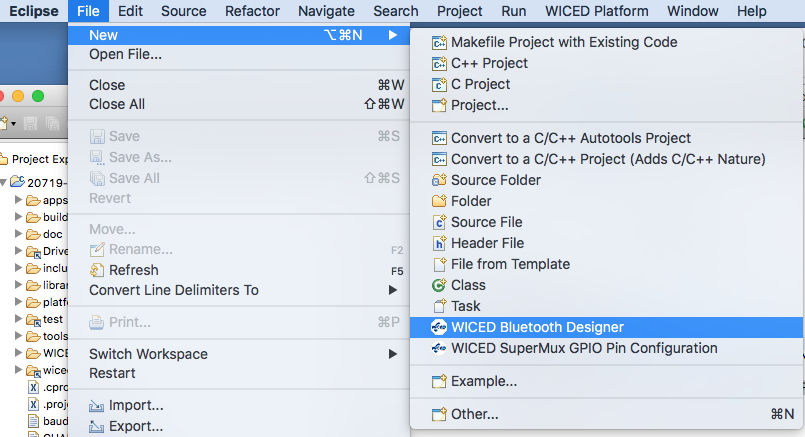
The Serial Port Profile specifies all the protocols and procedures required to setup, discover and connect two virtual serial port over an RFCOMM connection.

# WICED Bluetooth Designer

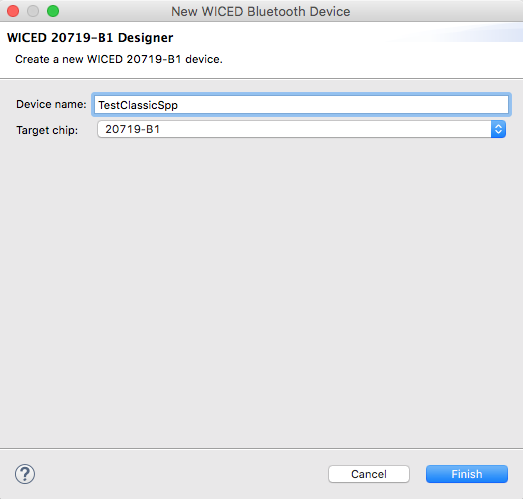
As with BLE, WICED Bluetooth Designer can be used to help you create a WICED Bluetooth Classic Project. Specifically, it will help you:

* Make a template project with all the required files
* Create the SDP Database
* Create a Make Target

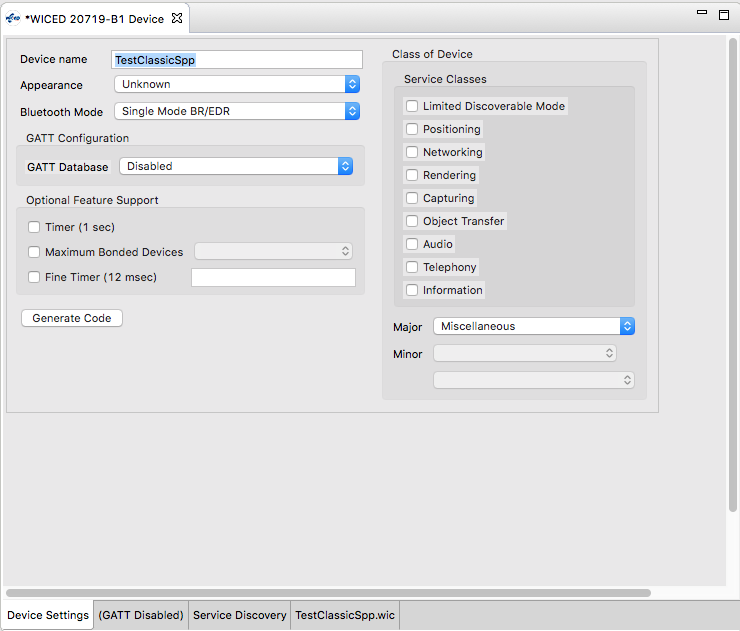
To run the tool go to File 🡪 New 🡪 WICED Bluetooth Designer



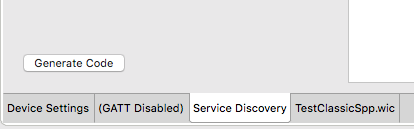
Give your project a name, in this case TestClassicSpp.



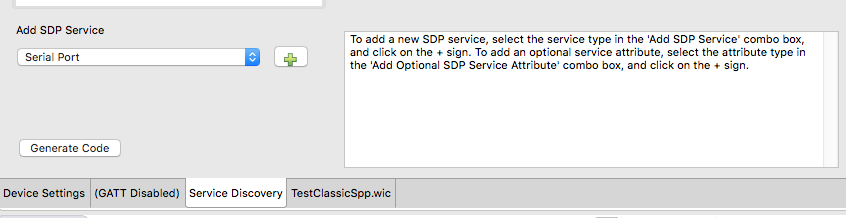
The default setting is a Single Mode LE, however, we want a Bluetooth Classic BR/EDR project, so change the Bluetooth Mode to “Single Mode BR/EDR”



The biggest benefit of the Bluetooth Designer is helping you build the “Service Discovery” database. Click on the Service Discovery Tab.



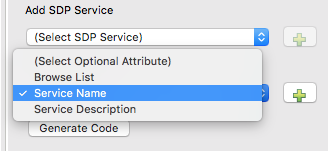
The SDP database starts with nothing in it. To add Services to the database, click on the drop-down menu title “Add SDP Service”, pick out the Service you want (in the case Serial Port) then press the “+”

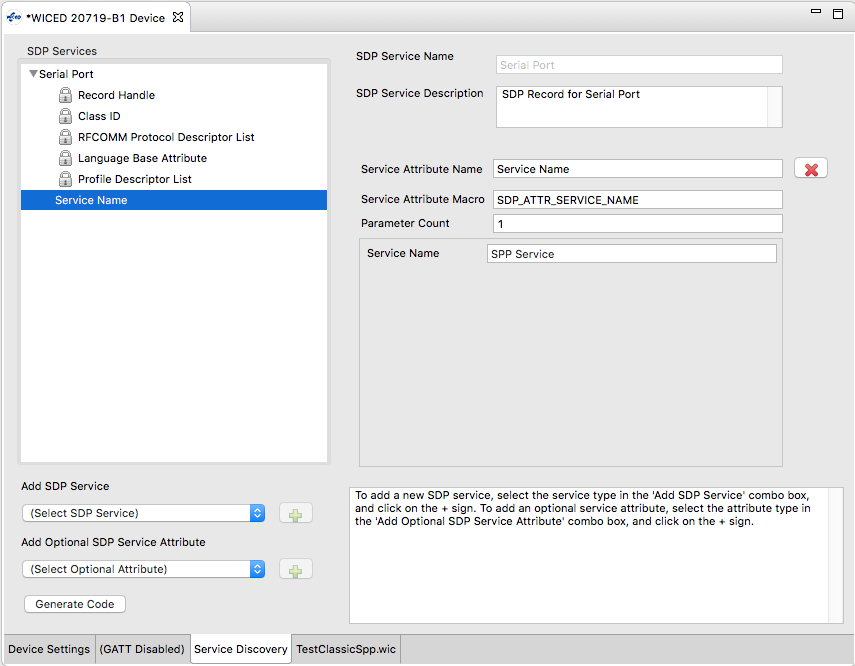


Now the SDP Database (which is in the window titled SDP Services) has the Serial Port and all the other stuff you need to go with it. To see all of that you need to click on the Arrow to the left of “Serial Port” to expand the Attributes.

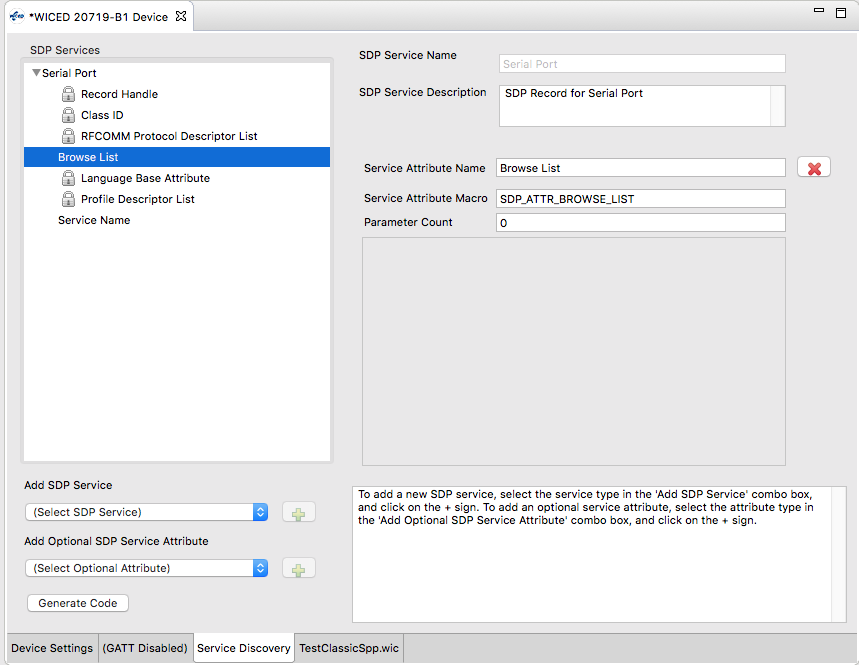


By default the Service Name and Browse List are not included in the Option SDP Service Attributes.. To add them, select from them on the “Add Optional SDL Service Attribute” menu and press the “+”. Here I add the “Service Name”

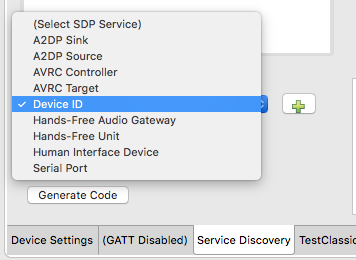
  
Then I change the name to “SPP Service” by typing in the “Service Name” text box.



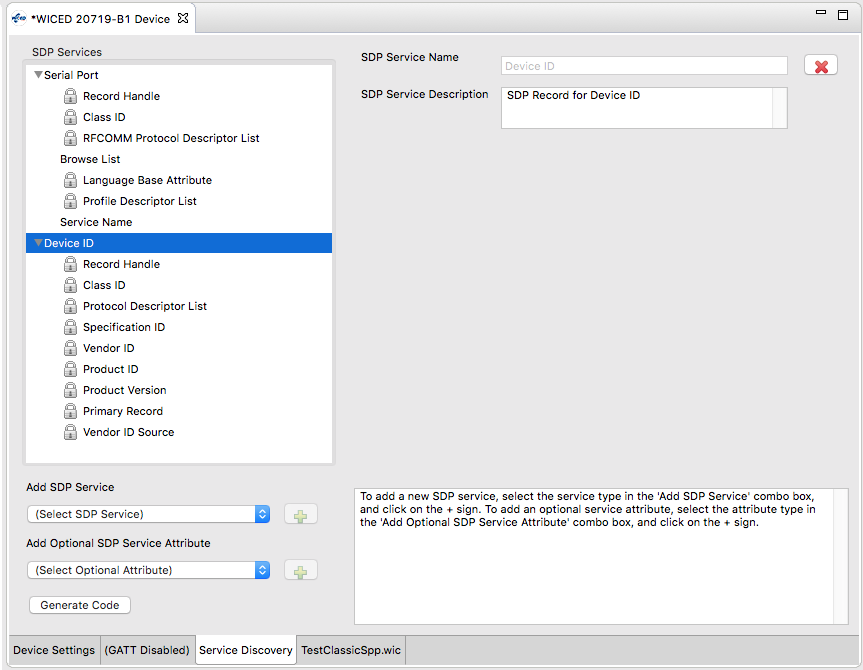
To add the Service to the Browse List, select “Browse List” from the Optional SDL Service Attribute List and press “+”



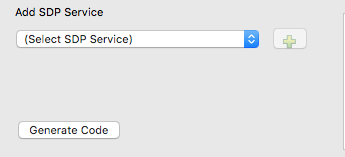
The next SDP Service that I add to the Database is the Device Name ID Service.



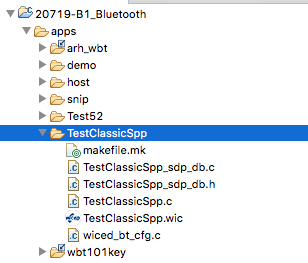
After it is added, you can see all of the Attributes that go with it.



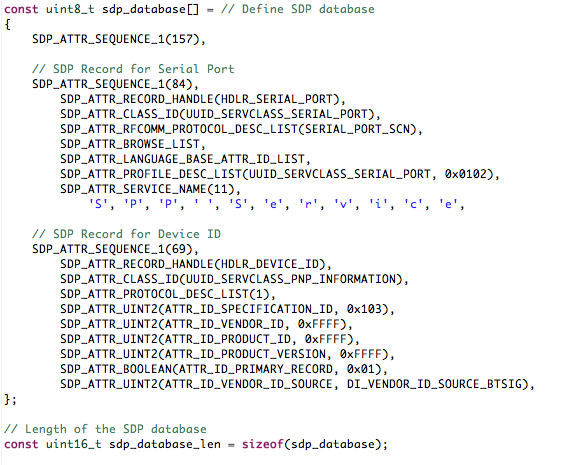
The last step in the process of making a project is to press the Generate Code which will build you project, create a make target, and customize all of the C-Code.



Here is the resulting workspace.



If you look in TestClassicSpp\_sdp\_db.c you will find the array of uint8\_t’s that represent the database



## WICED Studio 6.2 – BT Designer Bugs

In WICED Studio 6.2 the BT Designer generates projects with missing includes. To get rid of these warnings you will need to add

* #include “wiced\_bt\_stack.h”
* #include “wiced\_bt\_sdp.h”
* #include “wiced\_hal\_wdog.h”
* #include “wiced\_bt\_app\_common.h”

# WICED Bluetooth Stack Events

The Stack generates Events based on what is happening in the Bluetooth world. After an event is created, the Stack will call the callback function which you registered when you turned on the Stack. Your callback firmware must look at the event code and the event parameter and take the appropriate action.

For your Basic Application these are the relevant BTM Events:

|  |  |
| --- | --- |
| **Event** | **Description** |
| BTM\_ENABLED\_EVT | When the Stack has everything going. This event data will tell if you it happened with WICED\_SUCCESS or !WICED\_SUCCESS. |
| BTM\_SECURITY\_REQUEST\_EVT | ? |
| BTM\_PAIRING\_IO\_CAPABILITIES\_BR\_EDR\_REQUEST\_EVT | The Stack is asking what IO capabilities this device has (Display, Keyboard etc) you need to update the structure sent to you in the event data. |
| BTM\_PAIRING\_COMPLETE\_EVT | The Stack is informing you that you are now paired. |
| BTM\_ENCRYPTION\_STATUS\_EVT | The Stack is informing you that the link is now encrypted… or not |
| BTM\_PAIRED\_DEVICE\_LINK\_KEYS\_REQUEST\_EVT | The Stack is asking you find and return the link key for the BDADDR that was sent in the event data. |
| BTM\_USER\_CONFIRMATION\_REQUEST\_EVT | The Stack is asking you to ask the user if the PIN you are displaying matches the PIN from the other side. |
| BTM\_LOCAL\_IDENTITY\_KEYS\_REQUEST\_EVT | The Stack is asking you to read the local identify keys from the NVRAM and return them to the Stack. |
| BTM\_PAIRING\_IO\_CAPABILITIES\_BR\_EDR\_RESPONSE\_EVT | The Stack is informing you of the I/O capabilities of the other side of the connection. |
| BTM\_PAIRED\_DEVICE\_LINK\_KEYS\_UPDATE\_EVT | The Stack is asking your firmware to store the BDADDR/Link Keys (which are passed in the event data) |

# WICED Classic Bluetooth Firmware Architecture

WICED Bluetooth Designer will create a skeleton of the firmware that you need start building your device. The skeleton includes a file named <appname>.c which contains

* The initialization functions application\_start function which is the entry point for your firmware and <appname>\_app\_init which provides a place for you to get the Bluetooth stuff going
* A template BTM event handler function

A dotH dotC pair of files called <appname>\_sdp\_db.h that contain the #defines for the SDP database the uint8\_t structure holding the actual database.

A file wiced\_bt\_config.h which contains all the basic Bluetooth configuration settings to get the stack going.

To make your project work you need to add the capability to:

* Handle Pairing
* Handle Bonding
* Support the Serial Port Profile

## Initialization Functions

The application\_start function is the entry point of your firmware. That function must:

* Initialize the memory pools (just like BLE)
* Configure the debugging UART
* Call wiced\_bt\_stack\_init with the event handler to start the stack

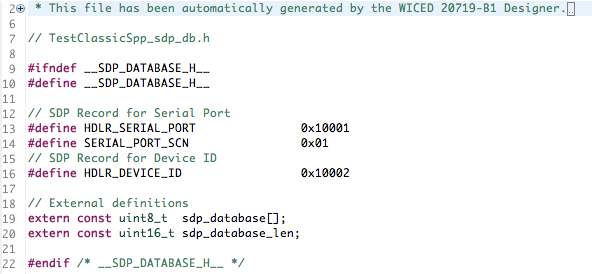
The <appname>\_app\_init function is created for you as a place to initialize your project. It is called in the BTM event handler after the stack starts.

* Makes your device pairable
* Initializes the SDP database
* Makes your device connectable (turns on Paging)
* Makes your device discoverable (turns on Inquiry)

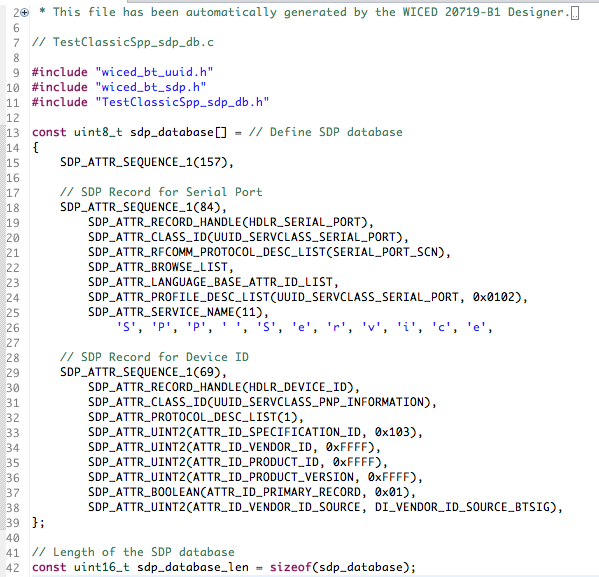
## SDP Database

The Service Discovery Database is created for you based on the configuration settings in the BT Designer. BT Designer creates the dotC and dotH files <appname>\_sdp\_db

The file <appname>\_sdp\_db.h



The file <appname>\_sdp\_db.c simply contains the two Service Records that we defined in the BT Designer, the SPP and the Device Info.



The only firmware action required by your application is to register your database which is inserted into the function <appname>\_app\_init for you automatically by the BT Designer.



## Handle Pairing

The BTM events involved in Pairing are:

* BTM\_PIN\_REQUEST\_EVT
* BTM\_PAIRING\_IO\_CAPABILITIES\_BR\_EDR\_RESPONSE\_EVT
* BTM\_PAIRING\_IO\_CAPABILITIES\_BR\_EDR\_REQUEST\_EVT
* BTM\_USER\_CONFIRMATION\_REQUEST\_EVT
* BTM\_PAIRING\_COMPLETE\_EVT

When you get the BTM\_PIN\_REQUEST\_EVT you should respond with a PIN code. The BT Designer does not put this code in the template for you. The pin code is typically four ASCII digits.



And you need to handle the case by responding with the PIN Code.



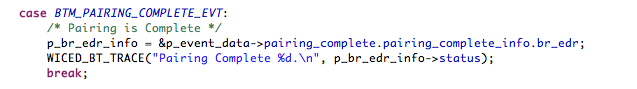
When the Master attempts to Pair with you, it sends its I/O capabilities. When you get the event you can decide what to do, including nothing. In this case, we just print out the I/O capabilities. By default, the BT Designer does not include this event in the template.



When you get the event asking for your I/O Capabilities, you need to respond by changing the event data. By default, the BT Designer gives you an incomplete filling of the structure. You need to add



When the pairing process is complete you can print a message, or more likely initialize some variable for that session.



## Handle Bonding

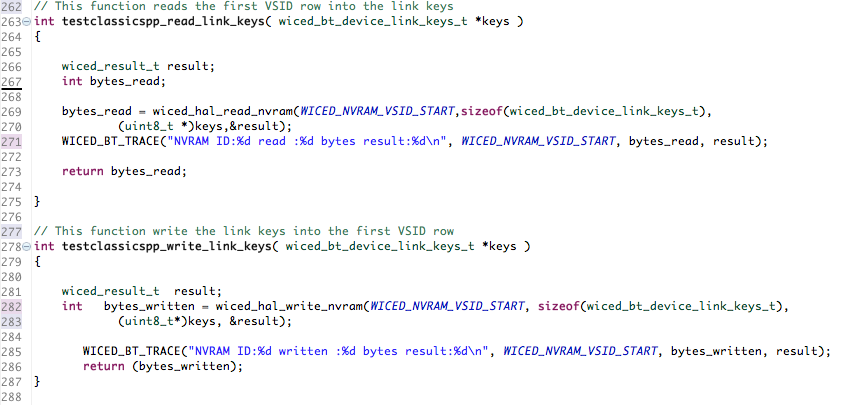
To handle Bonding, you need to act on the BTM Events that request you to find the saved link key for a specific BD Address (BTM\_PAIRED\_DEVICE\_LINK\_KEYS\_REQUEST\_EVT) and to save the Link Key when it is created or updated for a specific address (BTM\_PAIRED\_DEVICE\_LINK\_KEYS\_UPDATE\_EVT)

This is best done by reserving a block of the NVRAM VS IDs to hold the key/BDADDR tuple. The VSID is just a row number in the NVRAM. Each Volatile Sector in the NVRAM is 255 bytes long. Each WICED Device has #defines for the WICED\_NVRAM\_VSID\_START and WICED\_NVRAM\_VSID\_END in wiced\_hal\_nvram.h

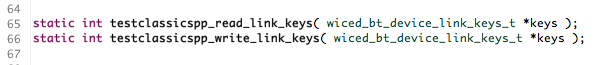
The BT Designer provides a template for the keys request event. In order to use it change the #if to 1.. You also need to provide the function <appname>\_read\_link\_keys



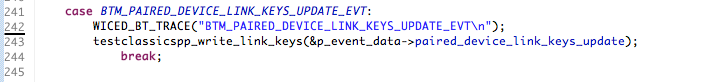
You need to make the read\_link\_keys and write\_link\_keys function. They both support only one set of saved link keys, and they use an entire VSID row.



After you add these two function you need to make a forward declaration for both of them at the top of the file.



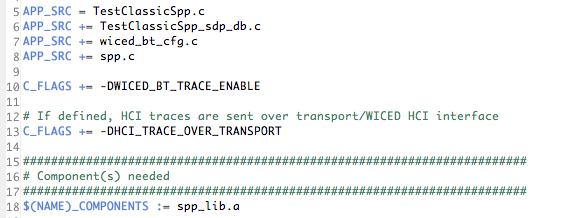
The BT Designer does not provide a template for the save keys event. So, you need to add the case for that event, which simply calls the write function you already created.



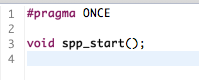
## Serial Port Profile

The WICED Bluetooth SDK contains all the code to implement the SPP server. To make it work you need to initialize the spp server and provide callbacks for starting and stopping connection, and receiving data. After you have created your SPP project with the BT Designer tool, you can open the snip.bt.spp example to copy the additional blocks of code that you need. Typically, I create a separate files spp.h and spp.c to handle all the SPP server functionality.

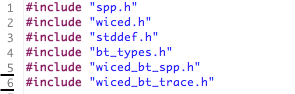
This leads you to two changes that need to happen to the makefile.mk. First you need to add the spp\_lib.a so that you can link against the SPP functions. And, when you add new files to your project you need to add them to the file makefile.mk.



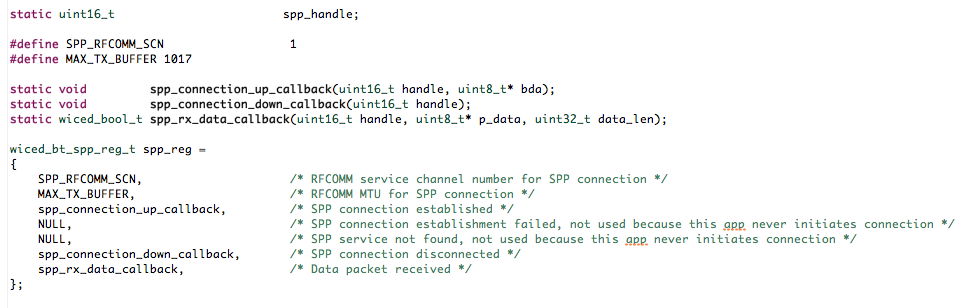
In spp.h I provide a function prototype for a public interface for the server.



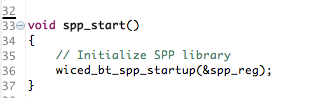
In spp.c you need add #defines to get all the required functions



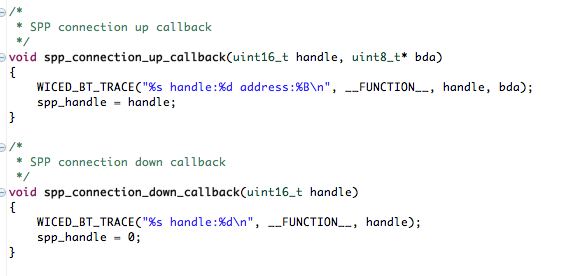
Then you declare a variable called spp\_handle to hold the current handle of the SPP connection.. You also need to make forward declarations for the SPP handler functions.

The structure wiced\_bt\_spp\_reg\_t holds all of the configuration information for the SPP Server. 

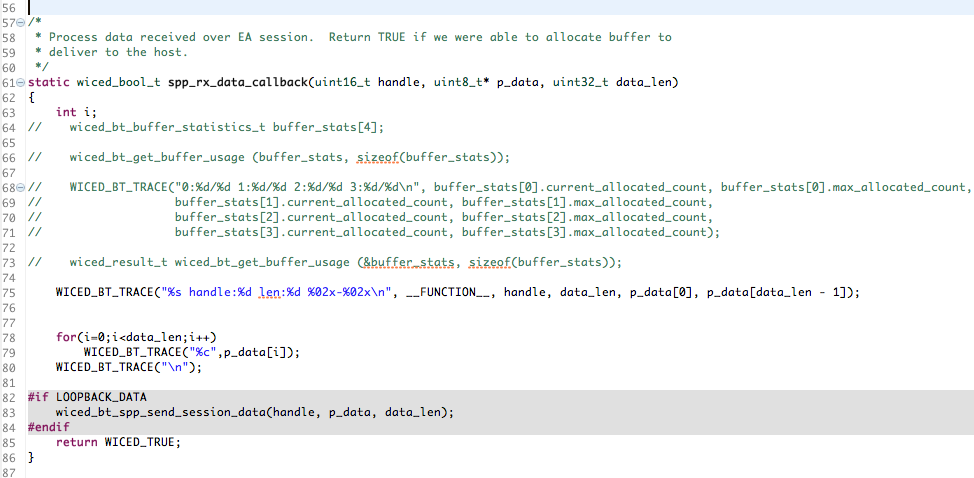
To startup the SPP server call the startup function with the configuration you defined above.



The connection up and down callback simply information via the BT Trace and set/unset a global variable that keeps track of the SPP handle.



When you receive data just dump it out onto the screen.



The last thing that you need to do is initialize the SPP server in your <appname>\_app\_init function



# Exercises

* 1. Create a Serial Port Profile Project

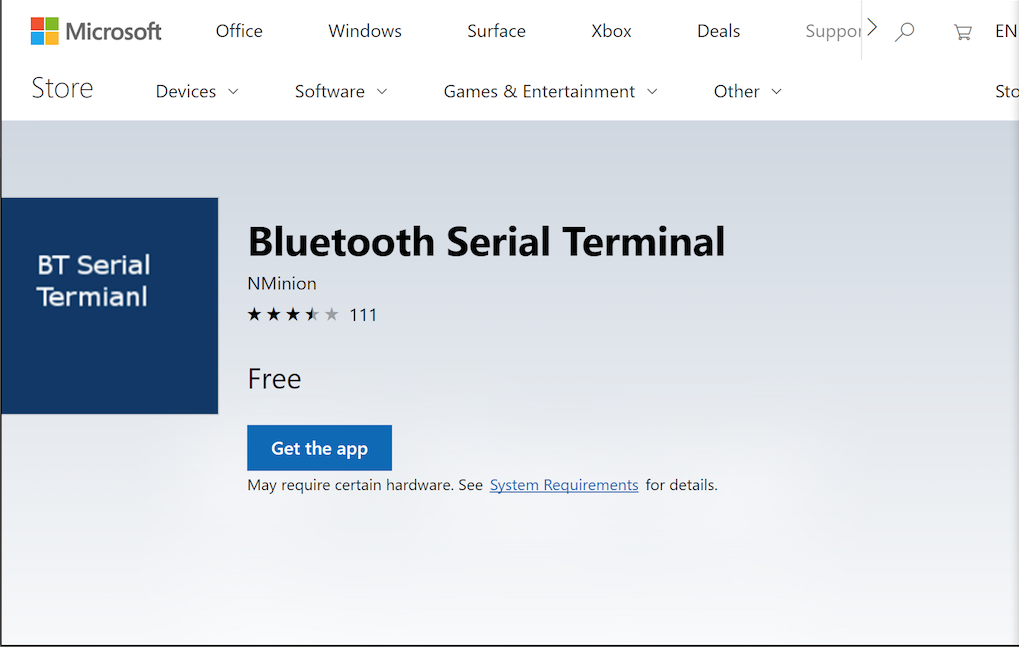
For this example, you will need to do:

1. Create a new project using the BT Designer as documented in section 9.5
2. Update the project to handle pairing as documented in section 9.7.3
3. Update the project to handle bonding as documented in section 9.7.4
4. Update the project with the SPP configuration as documented in section 9.7.5

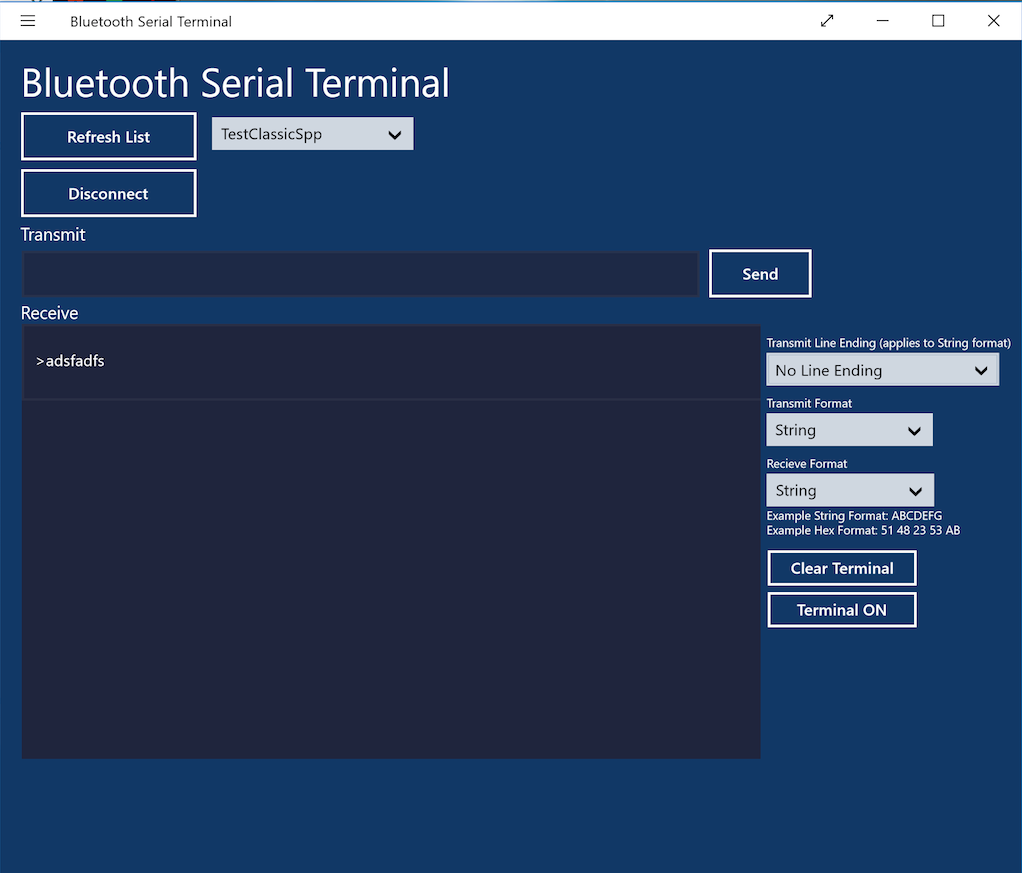
Once your project seems to be working, you can attach to it using Windows, Mac or Android.

**PC Instructions**

For Windows 10 the easiest thing to do is install the “Bluetooth Serial Terminal” from the Microsoft App Store.

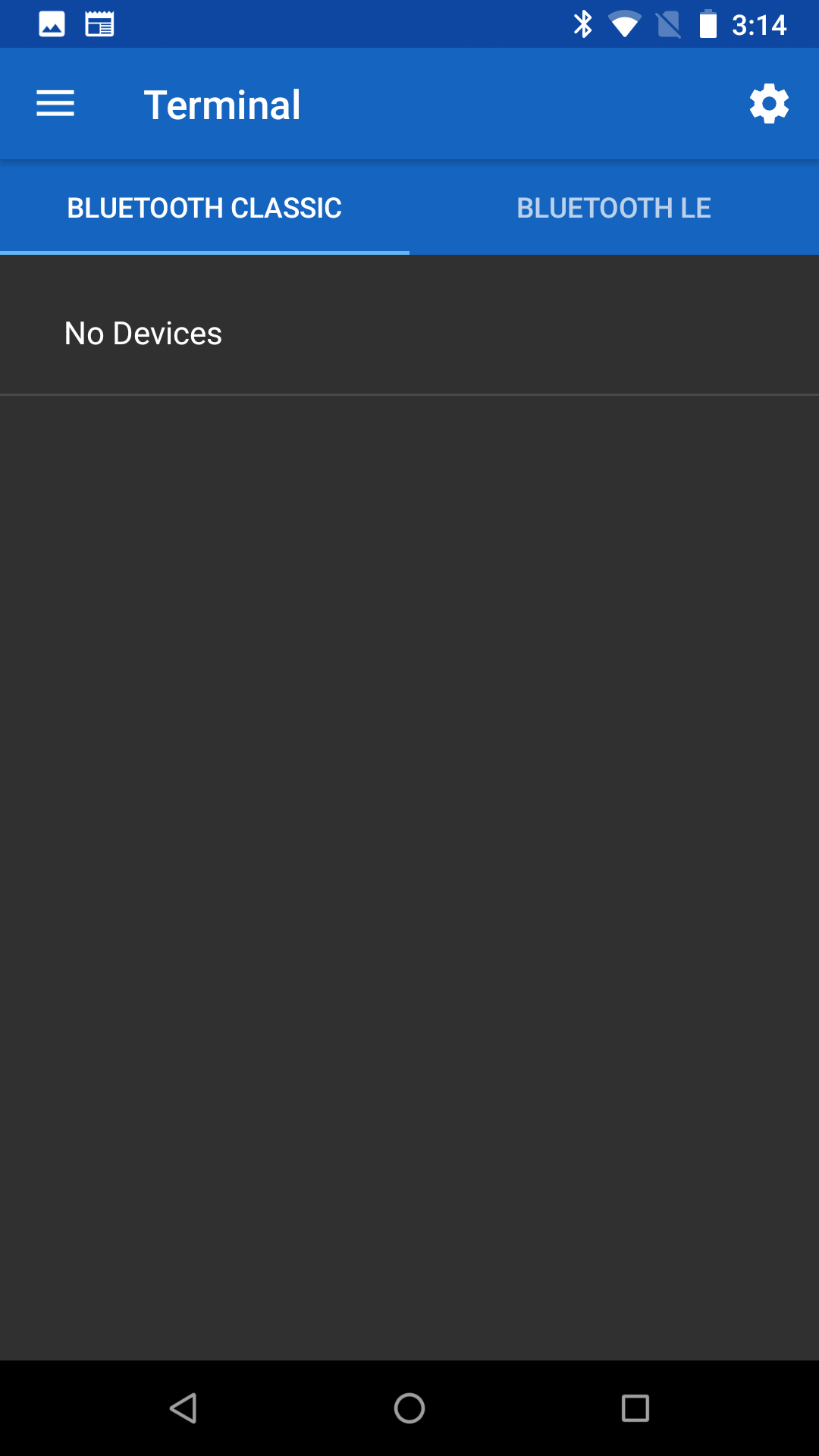


Once that is done you click “Refresh List” until you can see your project, in this case “TestClassicSpp”. Then Press “Connect”. Now you can type strings in the Transmit window to send to the WICED SPP Project.

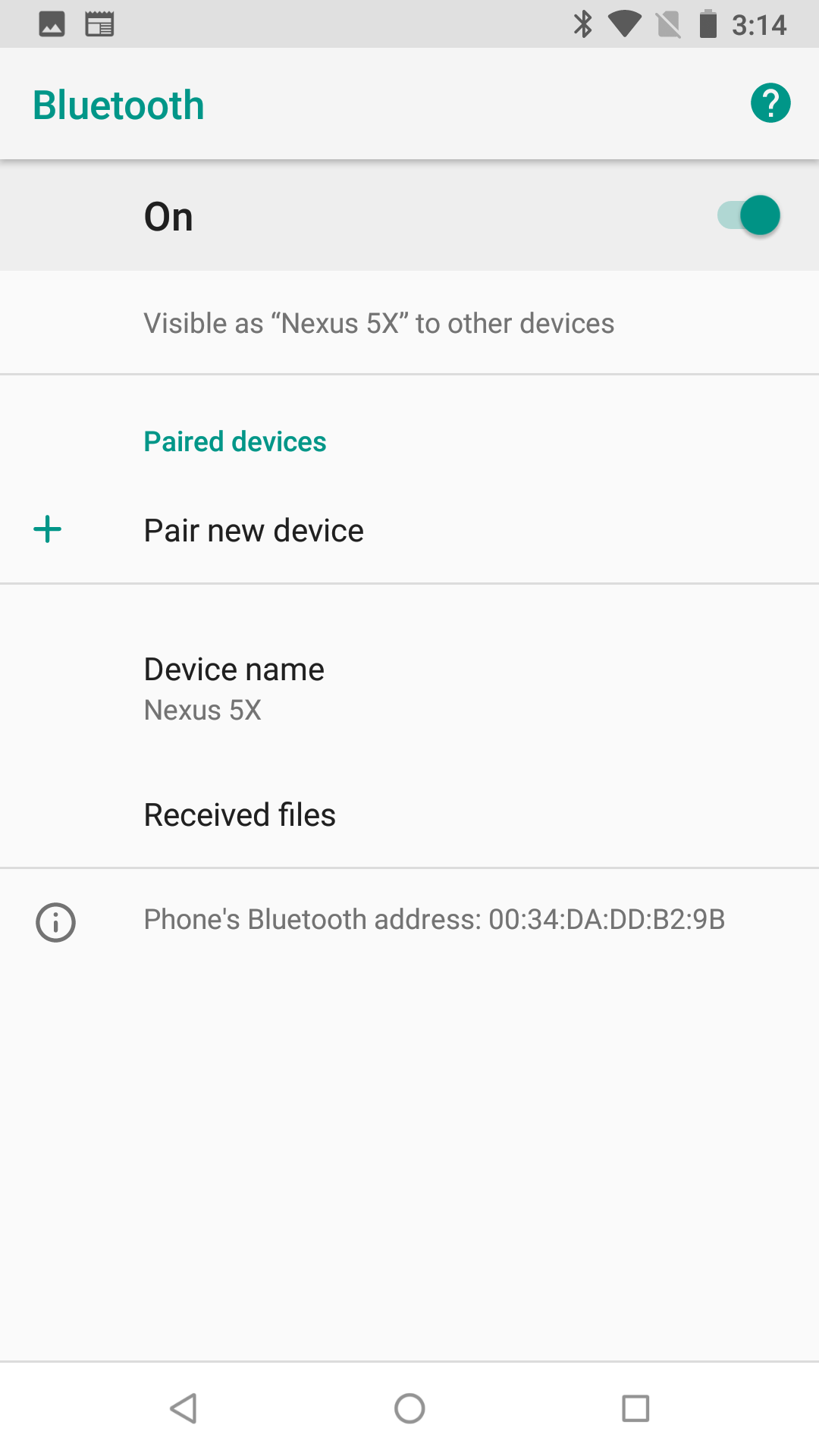
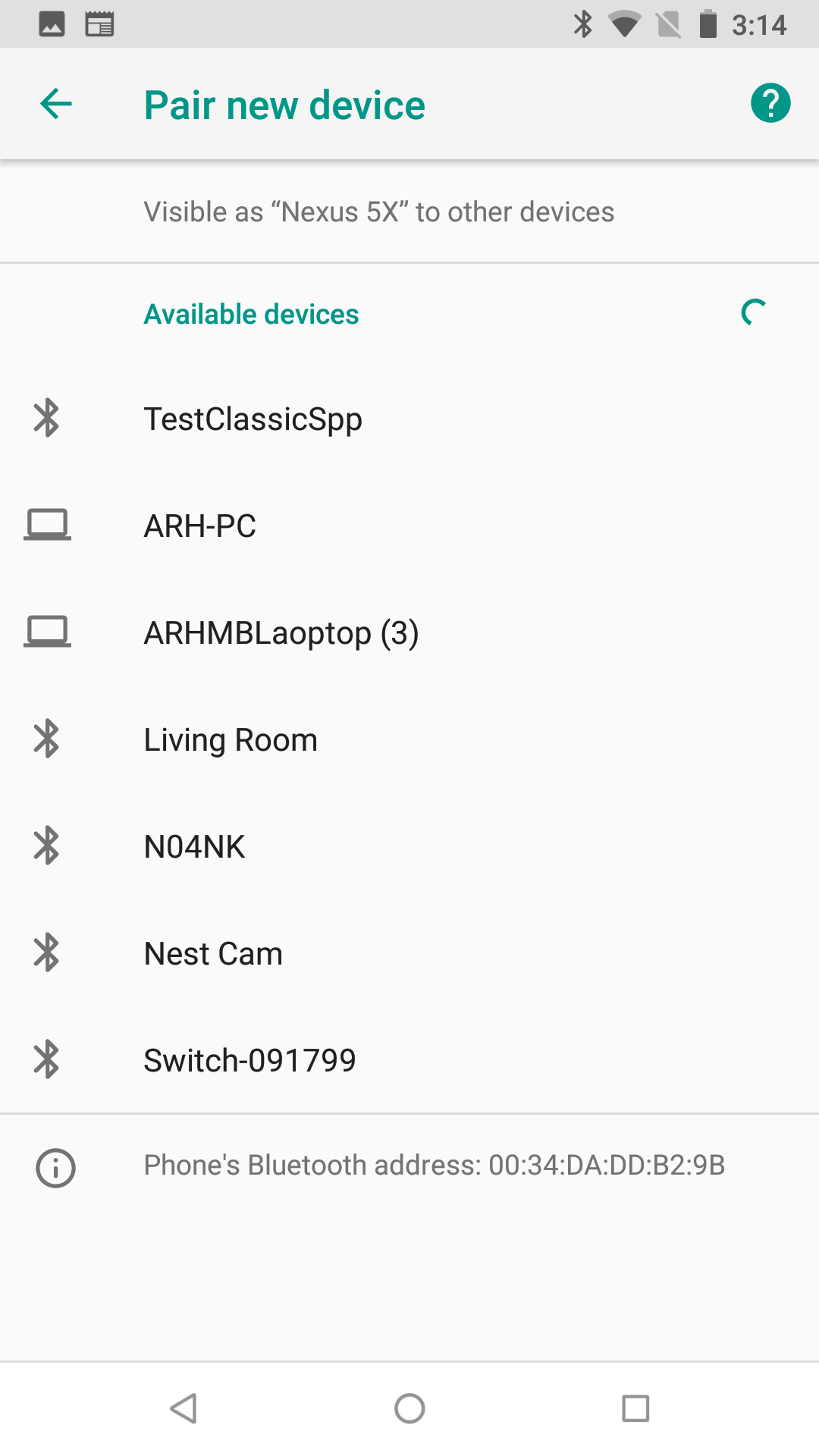
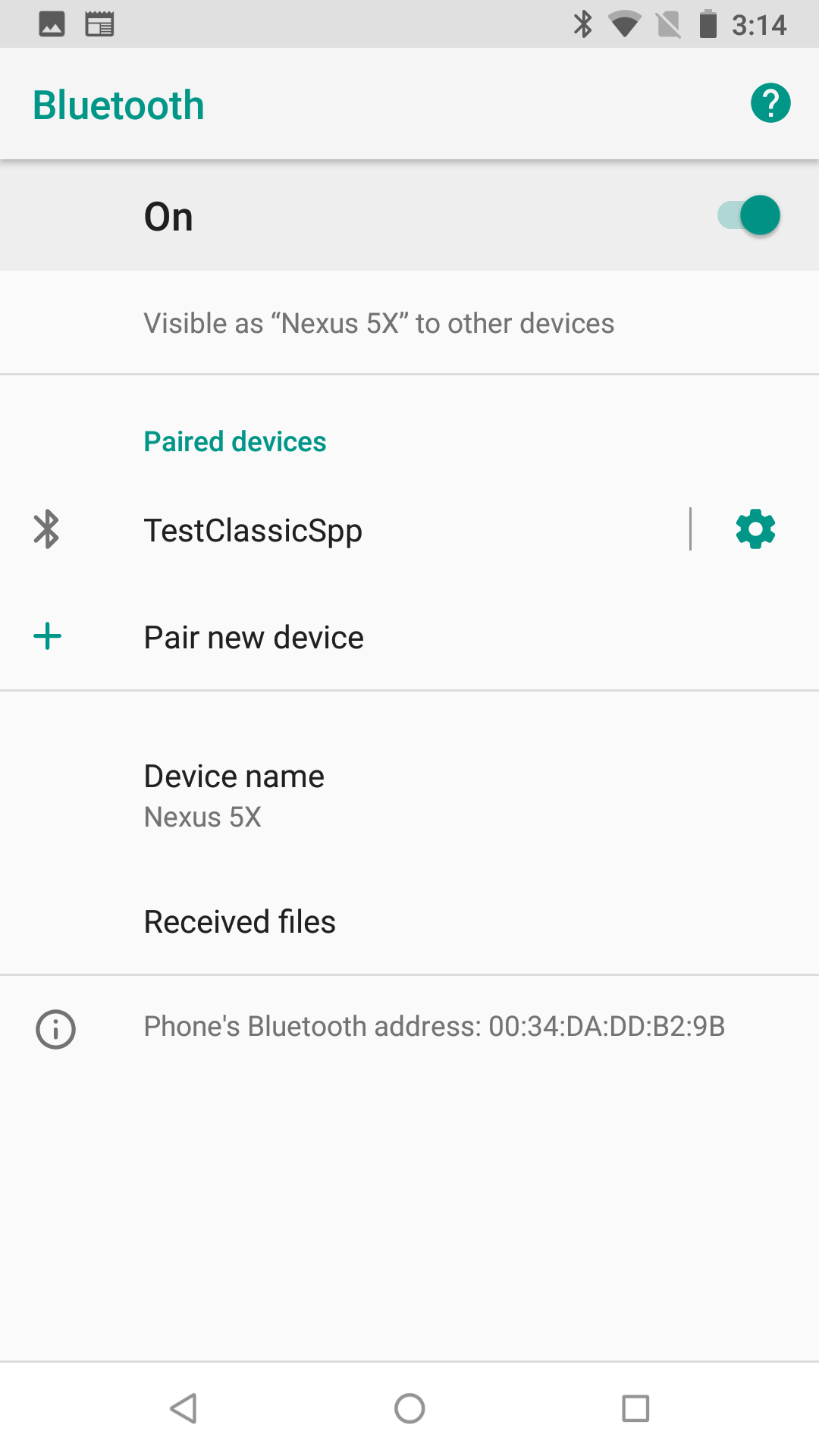


**Android Instructions**

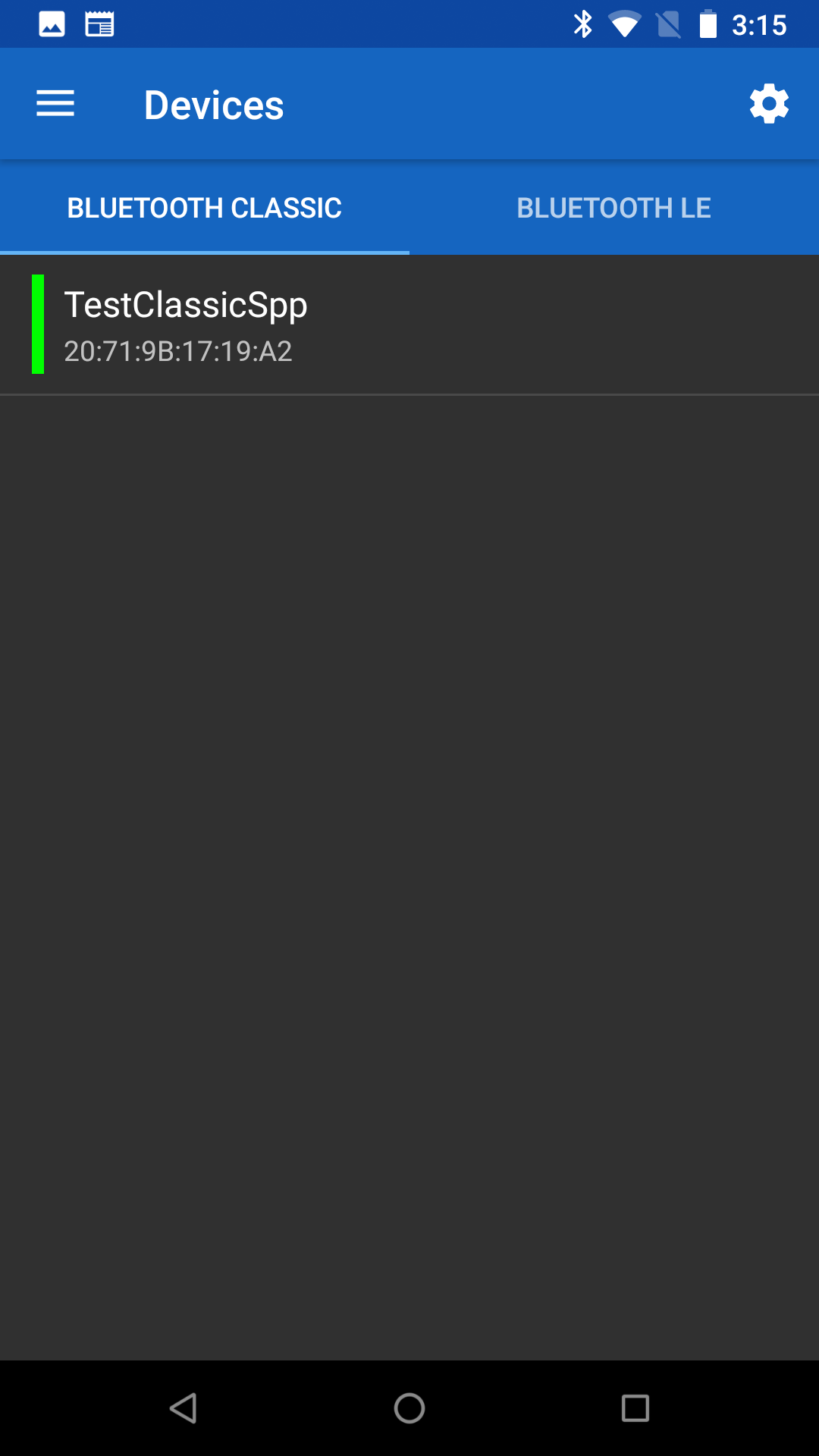
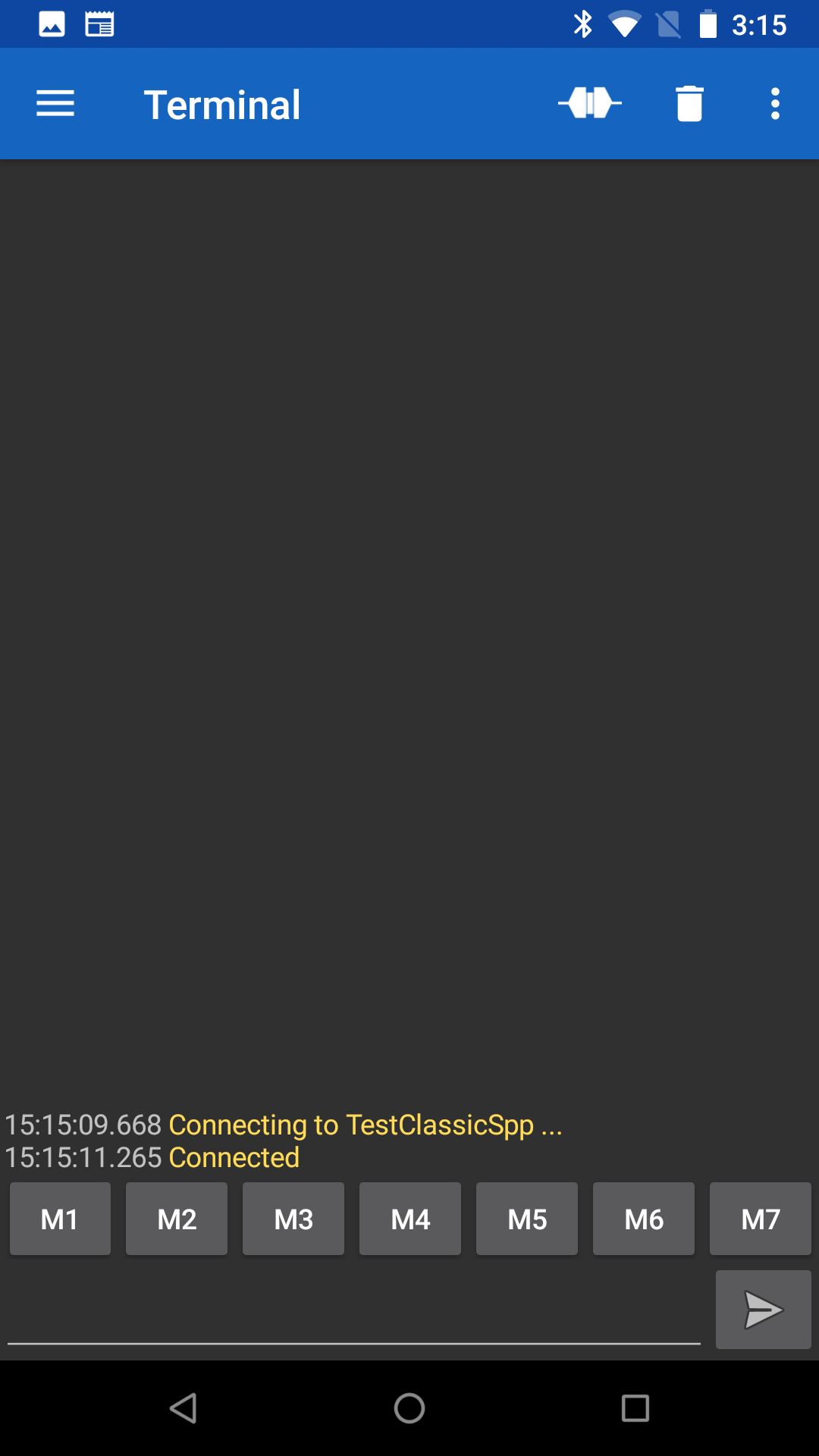
On an Android phone, you can install “Serial Bluetooth Terminal” from the Google Play Store. Install “Serial” from Decisive Tactics onto your Mac. You can get it in the App Store. When you run the App you will need to pair with your development kit. To do that press the thing in the upper left. Then Click on devices.

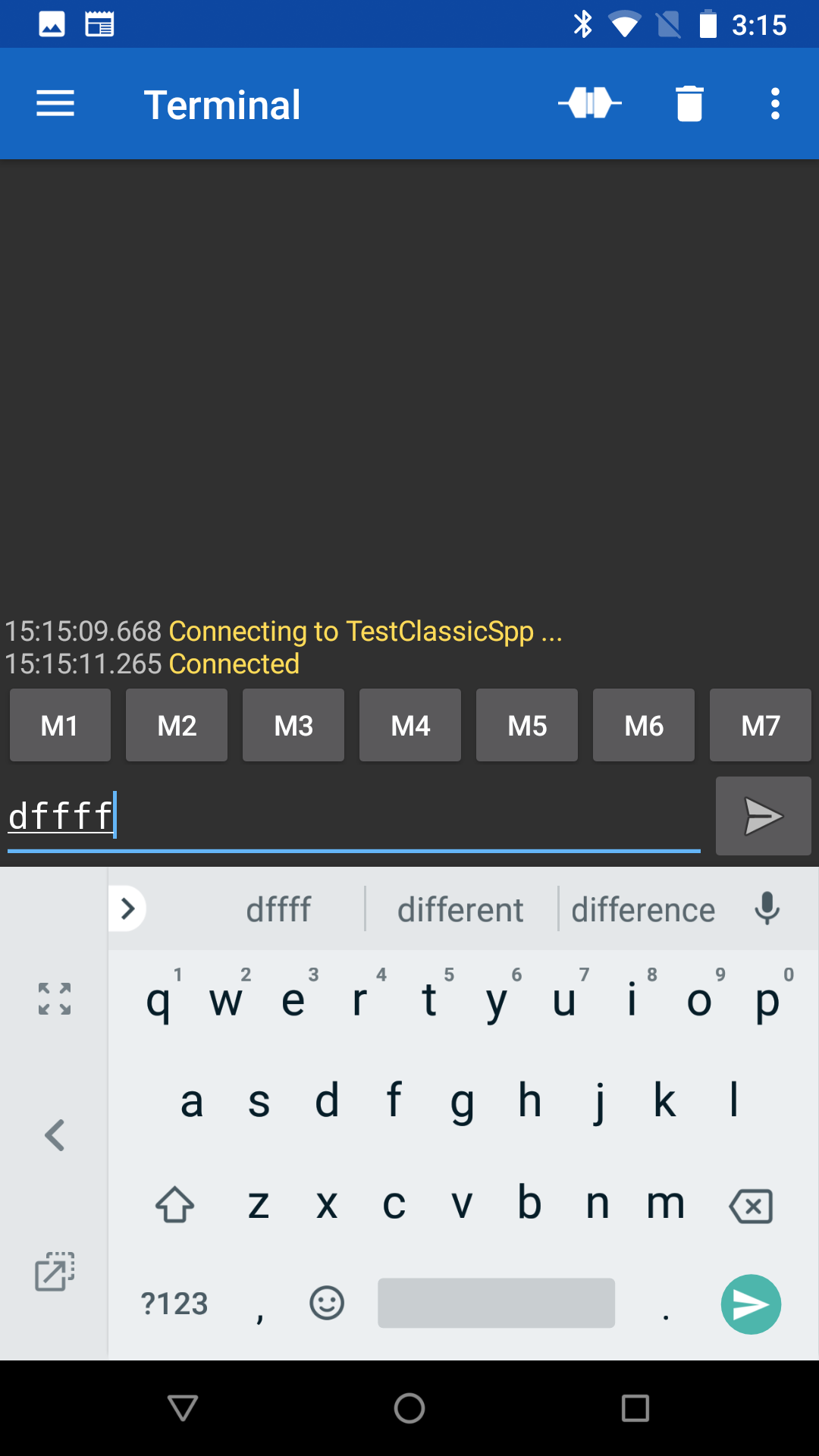
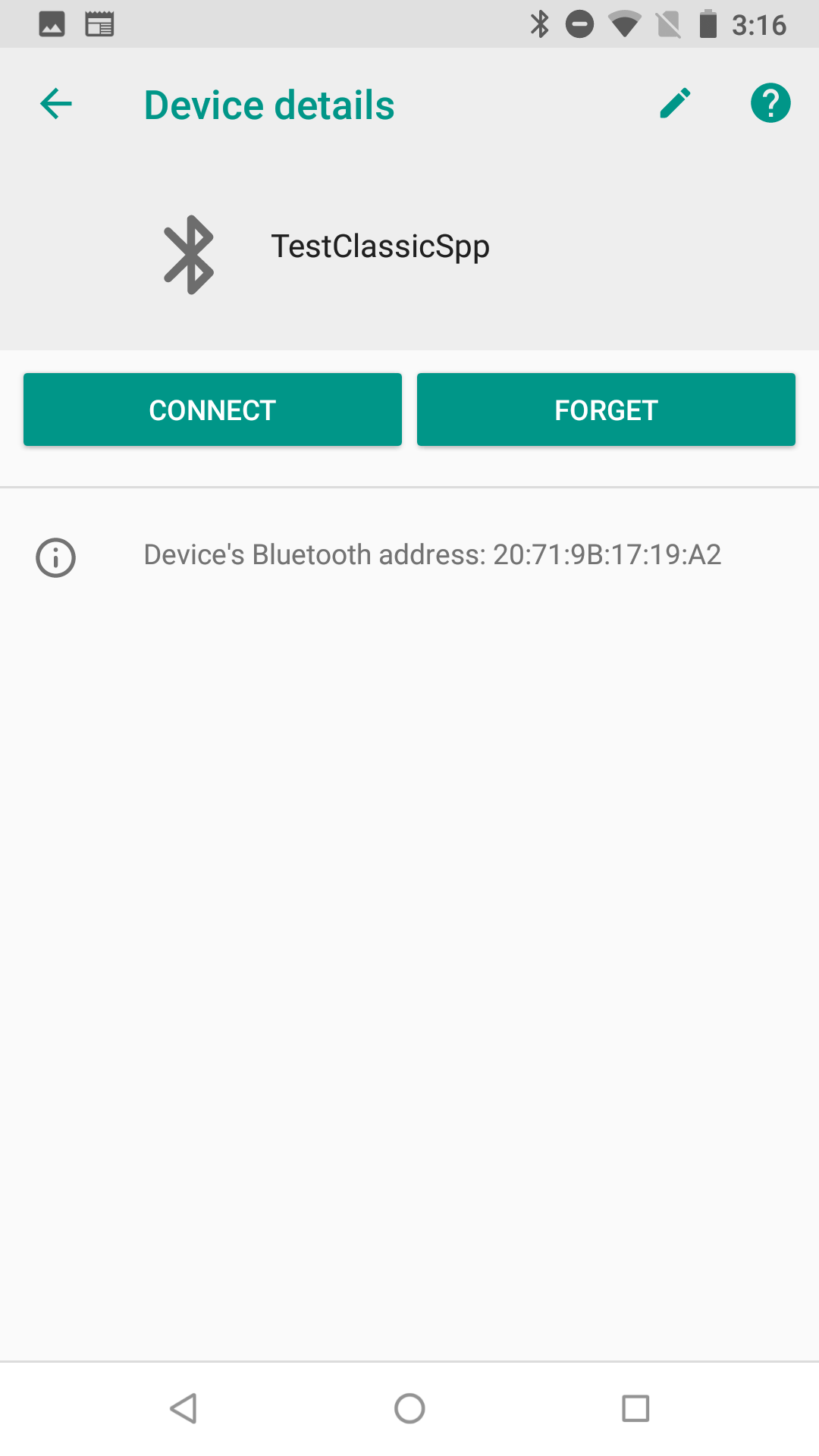
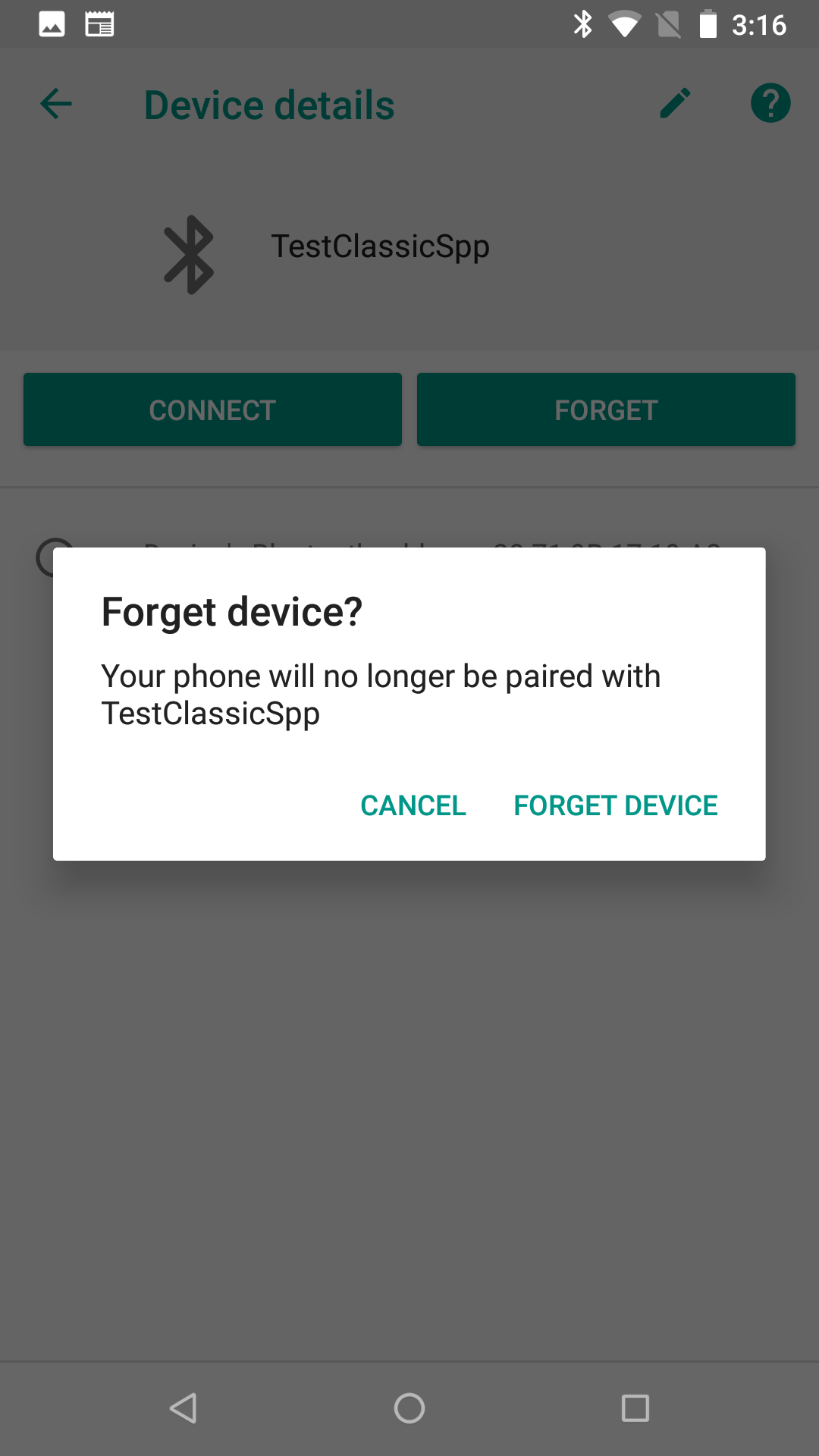
Click on “+ Pair new device”. Then find your development kit and add it.

After you are connected press the back arrow where you will see that your device is connected. Then press the back arrow again to see the blank terminal window. Next you next to press the plug icon to actually make a connection to your development kit.

Now you can send data to the SPP. When you press the plug again, it will disconnect. You can then go back to the Device screen and forget (aka delete the Bonding information) for your project.

**Mac Instructions**

Install the program “Serial” from the App Store.



Once you have programmed the development kit you need to connect to the board. In the Serial program choose File 🡪Open Bluetooth



Then find you project and press “Select”. This will pair to the development kit and open a window.



You will be asked to confirm the connection.



Once it is connected, everything you type will appear in the console window of the WICED Development kit. Below you can see that I typed “asdf”



To unpair your development kit, select the Bluetooth symbol and pick “Open Bluetooth Preferences”



Then click the “x”

And it will confirm that you want to remove the Bonding information from the Mac BT Stack.



* 1. Improve Security by Adding IO Capabilities
  2. Add Multiple Device Bonding Capability