Bluetooth KISS TNC

Build a Bluetooth KISS TNC with a Raspberry Pi and Dire Wolf.

WB2OSZ, May 2017

Cut the cord between your TNC and APRS/Packet application computer by using Bluetooth. Here is how.

For this example, we will use the Raspberry Pi, with the Jessie version of Raspbian. Other hardware or operating system versions might be a little different.

I happen to be using an RPi model 2 with a USB Bluetooth adapter. Using a model 3 or zero W, with the built in built-in Bluetooth, should be very similar if not identical.

Step 1: Install Dire Wolf version 1.5 or later.

The first step is get Dire Wolf running as explained here:

https://github.com/wb2osz/direwolf/raw/dev/doc/Raspberry-Pi-APRS.pdf

At the time this is being written, version 1.5 is still in the development stage. You need to use the "git checkout dev" command. Here is a quick cheat sheet. You will still need to read the document mentioned for initial configuration.

```
sudo apt-get update
sudo apt-get install libasound2-dev
git clone http://github.com/wb2osz/direwolf
cd direwolf
git checkout dev
make -j
sudo make install
make install-conf
cd
```

(Edit direwolf.conf file. If you are using a USB Audio Adapter, look for line with "# ADEVICE plughw:1,0" and remove the first character.)

Step 2: Install Bluetooth hardware and software

Next, we need a Bluetooth adapter. The Raspberry Pi model 3 and model zero W have it built in already. For older models, add a USB Bluetooth adapter such as these:

https://www.adafruit.com/product/1327

https://www.amazon.com/DayKit-Bluetooth-Adapter-Windows-Raspberry/dp/B01IM8YKPW/

Make sure the description mentions that it is compatible with the Raspberry Pi. Otherwise, you might end up with some chipset where a suitable driver is not available.

Install the bluez software.

```
sudo apt-get update
sudo apt-get install bluez bluez-tools
```

Step 3: Configure Bluetooth

Editing of the next 3 files must be done as "root." Use "sudo nano" or "sudo vi" or some other favorite text editor.

Edit /lib/systemd/system/bluetooth.service

Look for the line with ExecStart, and add "--compat" to the end as shown below. I have no idea why but I found this in about 5 different forums and couldn't get it to work properly without this.

```
ExecStart=/usr/lib/bluetooth/bluetoothd --compat
ExecStartPost=/usr/bin/sdptool add SP
ExecStartPost=/bin/hciconfig hci0 up piscan
```

The second line, above, adds the serial port profile. The other ensures that the device is up and allows this Bluetooth device to be found by others.

Create /etc/systemd/system/rfcomm.service

Create a new file containing:

```
[Unit]
Description=RFCOMM service
After=bluetooth.service
Requires=bluetooth.service

[Service]
ExecStart=/usr/bin/rfcomm watch hci0

[Install]
WantedBy=multi-user.target
```

Enable services

Type this so the services will start up automatically

```
sudo systemctl enable bluetooth
sudo systemctl enable rfcomm
```

Step 4: Reboot and initial tests

Reboot and verify that the services started up OK.

```
sudo reboot
```

Just type the first line below. The rest is part of the expected response. Be sure that it says “active (running)” rather than “inactive (dead)”.

```
systemctl status bluetooth

  bluetooth.service - Bluetooth service
   Loaded: loaded (/lib/systemd/system/bluetooth.service; enabled)
   Active: active (running) since …
```

Same for the “rfcomm” service.

```
systemctl status rfcomm

  rfcomm.service - RFCOMM service
   Loaded: loaded (/etc/systemd/system/rfcomm.service; enabled)
   Active: active (running) since …
```

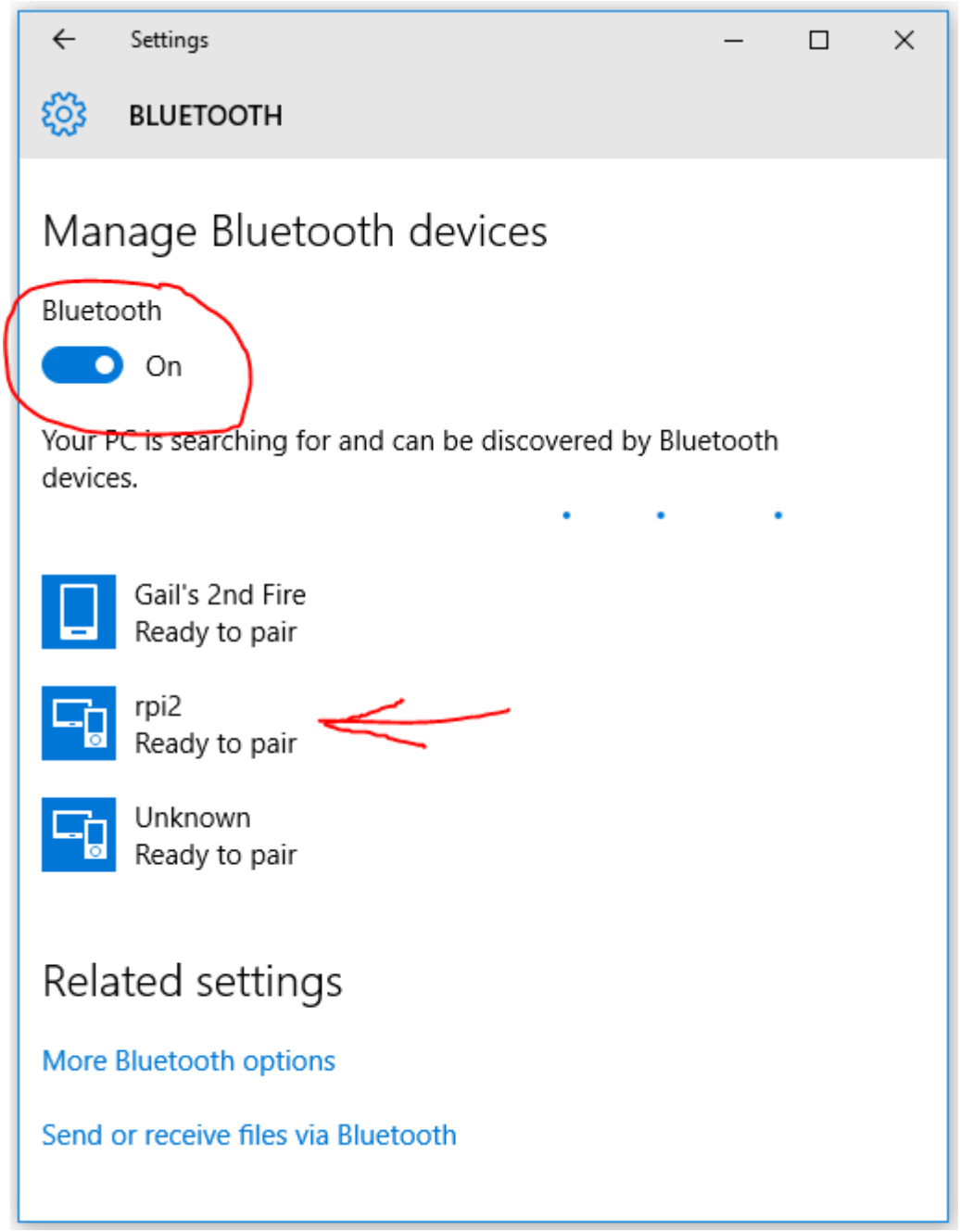
Do this to see what other Bluetooth enabled devices are nearby:

```
hcitool scan
```

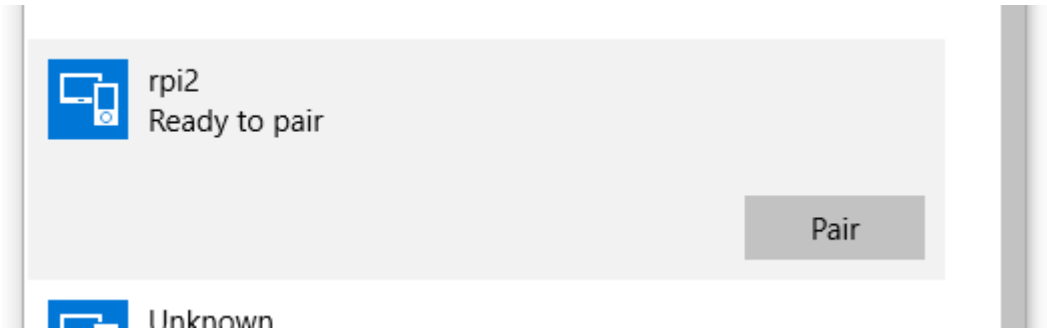
Step 5: Connect from another computer

Let's see if we can see the Raspberry Pi from another computer. In this example, we will be using Windows 10.

Click on the Windows Icon in the corner. Pick Settings → Devices → Bluetooth. Make sure Bluetooth is enabled and it should start scanning. In this case, we are looking for the host name "rpi2."

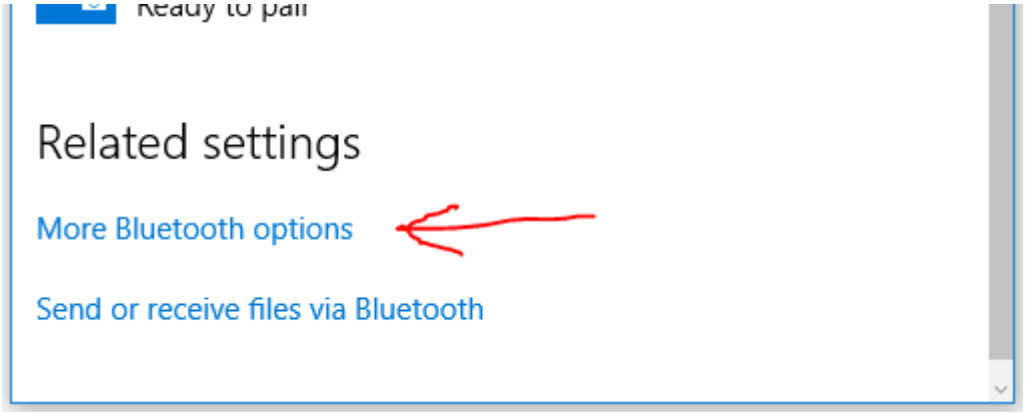


Select the name of your Raspberry Pi and a “Pair” button will appear. Click on it.

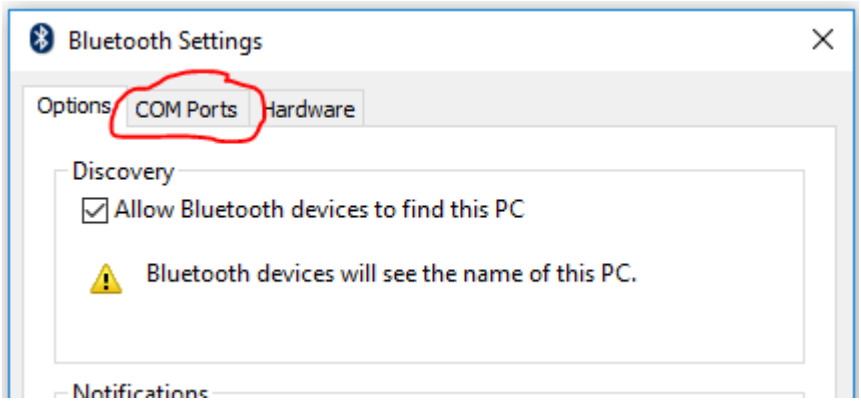


After a brief delay, it should now indicate “Paired” rather than “Ready to pair.”

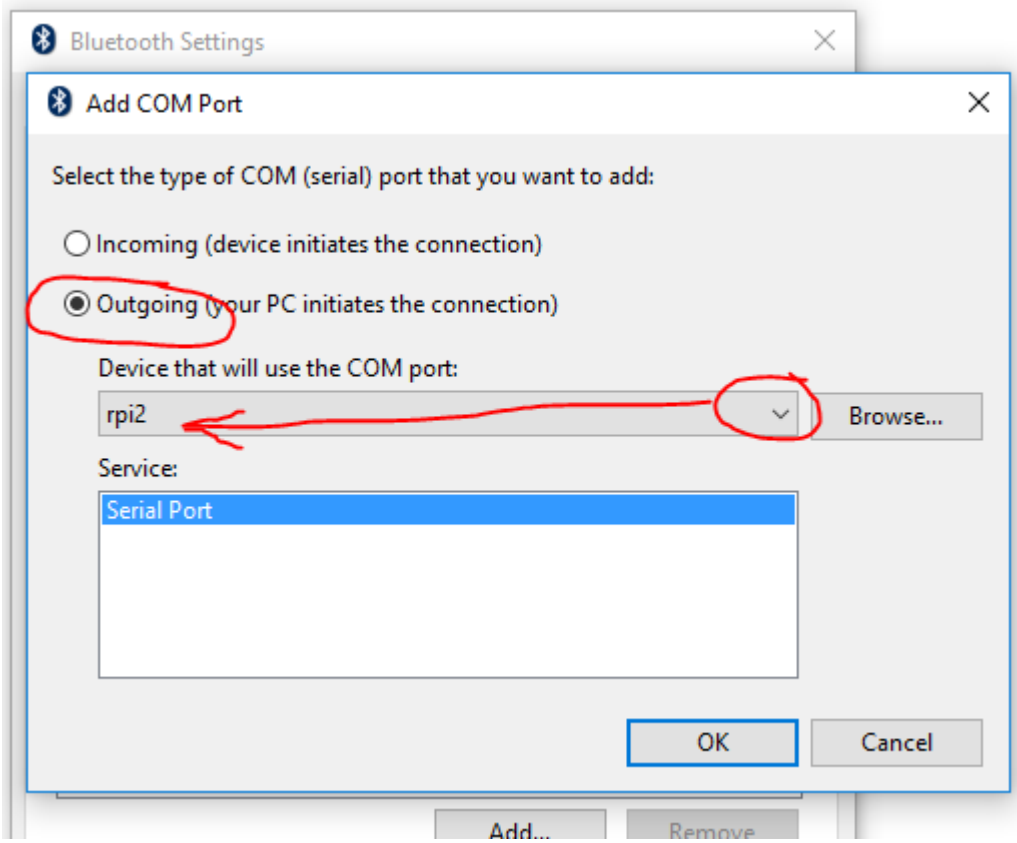
The next part is not at all intuitive. You would expect to double click or right click the host name to do something else with it. Instead, we need to go to “More Bluetooth options.”



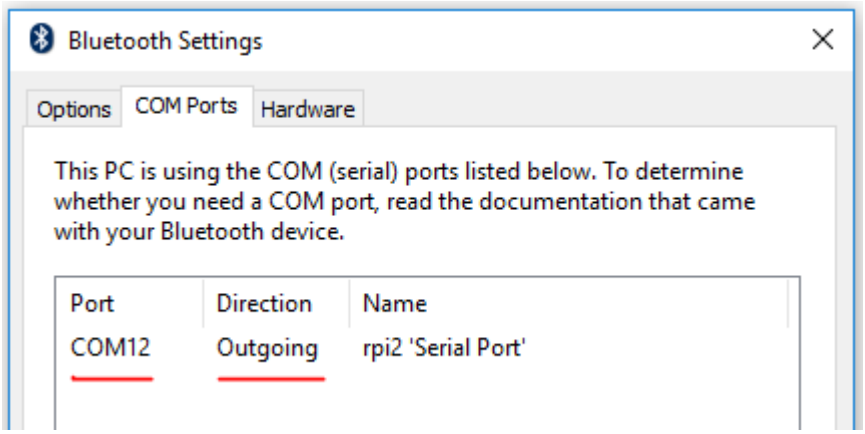
Select the “COM Ports” tab. Then click the “Add…” button which becomes available.



You should now see the “Add COM Port” dialog. Select Outgoing. Select the name of your Raspberry Pi from the drop down list. Finally click on OK.



If all goes well, you should have an **Outgoing** virtual serial port without the wire. You might also see one listed as "Incoming." We are not interested in that one.



Wish List: Is there some way to advertise it with a more descriptive name, such as "KISS TNC," rather than "Serial Port?"

A two way test

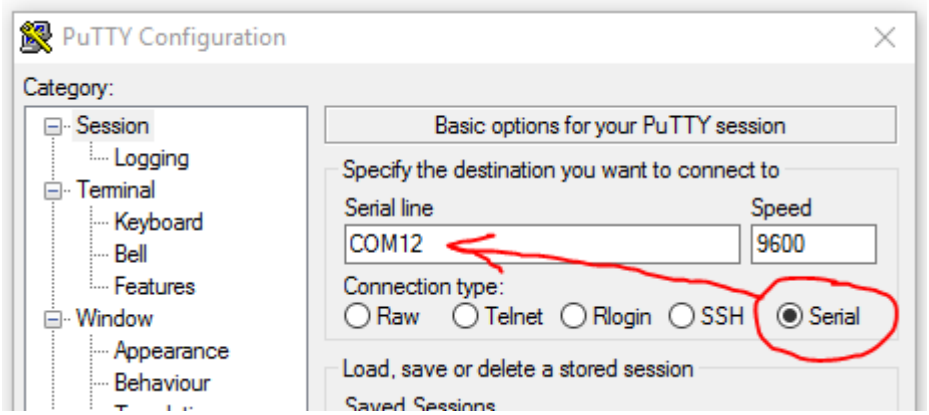
Back on the Raspberry Pi, type this:

```
ls -l /dev/rfc*
```

It should respond with “No such file or directory.” This is where it gets interesting.

We are going to run two terminal emulator programs and type back and forth between them.

On Windows, use PuTTY (http://www.putty.org/) or some other terminal application. Use the COM port number seen in the previous step. Click on the “Open” button near the bottom.



Back on the Raspberry Pi, type this again:

```
ls -l /dev/rfc*
```

This time there is a device called /dev/rfcomm0. Where did it come from? The act of an application opening COM12, on the Windows PC, caused a device called /dev/rfcomm0 to show up on the Raspberry Pi.

Now let’s connect a terminal emulator to this device. There are many alternatives available. I happened to use minicom:

```
minicom -D /dev/rfcomm0
```

Anything typed into one of the terminal windows will appear in the other.

Exit out of the terminal window on Linux. Observe that /dev/rfcomm0 still exists.

Exit out of the terminal window on Windows. Observe that /dev/rfcomm0 has disappeared.

This is an important point:

/dev/rfcomm0 will appear and disappear as an application opens and closes the virtual COM port on the other end.

Step 6: Use it as a KISS TNC

Dire Wolf has the ability to act as a KISS TNC through a pseudo terminal, TCP/IP over Ethernet / WiFi, or a serial port device. In this case we will use /dev/rfcomm0 in place of a traditional serial port. The problem is that it appears and disappears.

Dire Wolf version 1.5 has a new configuration option which will periodically check to see if the device is present rather than failing if the device is not there at start up time.

Just add this to your direwolf.conf configuration file:

```
SERIALKISSPOLL  /dev/rfcomm0
```

This will check periodically to see if the device exists and open it automatically as needed.

Application on Windows PC / Laptop

On the Windows PC end, configure your favorite APRS / packet application (APRSISCE, YAAC, SARTrack, etc.) to use serial port KISS connected to the virtual COM port.

Application on Android Phone / Tablet

NA7Q suggests another method that doesn't need the new SERIALKISS capability. Instead, use:

```
sudo rfcomm --raw watch /dev/rfcomm0 22 socat tcp4:127.0.0.1:8001 /dev/rfcomm0
```

This is how it works: When the /dev/rfcomm0 device is created, "socat" is started up automatically. It makes a two-way connection between this device and the TCP KISS port 8001.

The "--raw" is very important. It seems that some part of this likes to treat /dev/rfcomm0 like a terminal device and apply special processing to control characters. Raw mode means just push all the bytes through without changing anything.

Troubleshooting

Sometimes you might run into issues with the TNC and application talking to each other. You might need to look at what one is sending and what the other is receiving. They could differ if the operating system does something funny with control characters rather than passing everything along, unaltered, in "raw" mode.

A Quick Review of the KISS Frame

The complete official definition is here: http://www.ax25.net/kiss.aspx

Briefly, it is a way to move AX.25 frames between a dumb TNC and an application which provides the intelligence for the higher level communication protocol. Originally it was used with RS-232 serial ports but in more recent times we also use TCP/IP or Bluetooth.

A frame is simply a series of bytes with a special value indicating the beginning and end:

Frame Start (FEND)	**Channel (port) and command**	**AX.25 frame without the "flag" patterns or bit-stuffing which are used over the air.**	**Frame End (FEND)**
C0	Most often you will see 00 here for channel 0, data.	The first byte is the beginning of the destination address. You would expect to find: 82 thru B4 for 'A' thru 'Z' 60 thru 72 for '0' thru '9'	C0

The astute reader will wonder, "What happens if the frame data contains C0?" Read the protocol specification to solve this mystery.

Debugging Options

Most APRS / Packet Radio applications will have some sort of debugging options to reveal the communication with the TNC for troubleshooting. Depending on the application, it might or might not show the surrounding C0 at the start and end so you need to look out for that.

For direwolf, use one of these command line options:

-d k Dump KISS data for serial port or pseudo terminal.

-d n Dump KISS data for network TCP KISS.

The “decode_aprs” utility can be used to decipher the bytes you find here or in some other context. For example, in https://github.com/ge0rg/aprsdroid/issues/160 , we see this being discussed:

```
c0 00
82 a0 88 a4 62 66 e0
9c 82 6e a2 40 40 6e
ae 92 88 8a 62 40 62
ae 92 88 8a 64 40 63
03 f0
3d 34 36 31 30 2e 32 33 4e 2f 31 32 33 33 34 2e 32 36 57 3e 20 52 50 69
20 54 4e 43
c0
```

Is it valid? What does it mean?

From the earlier table, we recognize c0 as the beginning and end markers. The second byte, 00, means channel 0, data. The next byte, 82, indicates that the destination address starts with the letter “A.”

After that it gets more tedious. Many of us are not so good at doing tedious things but computers thrive on it. Put all those numbers into a file on a single line. Feed it into “decode_aprs” and we get:

```
c0 00 82 a0 88 a4 62 66 e0 9c 82 6e a2 40 40 6e ae 92 88 8a 62 40 62 ae 92 88
8a 64 40 63 03 f0 3d 34 36 31 30 2e 32 33 4e 2f 31 32 33 33 34 2e 32 36 57 3e
20 52 50 69 20 54 4e 43 c0

Removing KISS FEND characters at beginning and end.

--- KISS frame ---
  000:  00 82 a0 88 a4 62 66 e0 9c 82 6e a2 40 40 6e ae  .....bf...n.@@n.
  010:  92 88 8a 62 40 62 ae 92 88 8a 64 40 63 03 f0 3d  ...b@b....d@c..=
  020:  34 36 31 30 2e 32 33 4e 2f 31 32 33 33 34 2e 32  4610.23N/12334.2
  030:  36 57 3e 20 52 50 69 20 54 4e 43                 6W> RPi TNC

--- AX.25 frame ---
U frame UI: p/f=0, No layer 3 protocol implemented., length = 58
 dest    APDR13  0 c/r=1 res=3 last=0
 source  NA7Q    7 c/r=0 res=3 last=0
 digi 1  WIDE1   1   h=0 res=3 last=0
 digi 2  WIDE2   1   h=0 res=3 last=1
  000:  82 a0 88 a4 62 66 e0 9c 82 6e a2 40 40 6e ae 92  ....bf...n.@@n..
  010:  88 8a 62 40 62 ae 92 88 8a 64 40 63 03 f0 3d 34  ..b@b....d@c..=4
  020:  36 31 30 2e 32 33 4e 2f 31 32 33 33 34 2e 32 36  610.23N/12334.26
  030:  57 3e 20 52 50 69 20 54 4e 43                    W> RPi TNC
-------------------

NA7Q-7>APDR13,WIDE1-1,WIDE2-1:=4610.23N/12334.26W> RPi TNC
Position, normal car (side view), APRSdroid Android App http://aprsdroid.org/
N 46 10.2300, W 123 34.2600
 RPi TNC
```

Future Possibilities

- Require PIN for greater security.
- Use Bluetooth for audio too.