

Connect Raspberry Pi to a Wi-Fi network using command line

Firstly, connect a WiFi dongle to the USB port of the Raspberry Pi.

Open the Terminal and type the following commands-

```
sudo nano /etc/network/interfaces
```

In the nano text editor, you'll see something like this:

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp
```

That's the very basic configuration that governs your Pi's Ethernet connect (indicated by the eth0 portion). We need to add on a very minor bit to enable the Wi-Fi dongle. Use the arrow keys to move down below the existing entry and add the following lines:

```
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Once you've annotated the file, press CTRL+X to save the file and exit the nano editor.

At the prompt again, enter the following command:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Compare the contents of the file, if it exists, to the following code. If the file is empty, you can use this code to populate it. Take note of the commented lines (indicated by the # marks) to reference which variable you should use based on your current Wi-Fi node configuration.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev

update_config=1

network={

ssid="YOURSSID"

psk="YOURPASSWORD"

# Protocol type can be: RSN (for WP2) and WPA (for WPA1)

proto=WPA

# Key management type can be: WPA-PSK or WPA-EAP (Pre-Shared or Enterprise)

key_mgmt=WPA-PSK

# Pairwise can be CCMP or TKIP (for WPA2 or WPA1)

pairwise=TKIP

#Authorization option should be OPEN for both WPA1/WPA2 (in less commonly used are SHARED
and LEAP)

auth_alg=OPEN

}
```

When you're done editing the file, press CTRL+X to save and exit the document. Now is the time to unplug the Ethernet cable and plug in the Wi-Fi dongle.

At the command prompt, enter the following command

```
sudo reboot
```

When the device finishes rebooting, it should automatically connect to the Wi-Fi node. If for some reason it fails to appear on the network, you can always plug the Ethernet cable back in to double check the two files and the variables you altered.

Code for EM18 RFID reader

First, we need to open up the UART of the RaspBerry Pi.

UART pins are on pins 8 (TX), 10 (RX)

To search for available serial ports we use the command

```
dmesg | grep tty
```

The output is something like this

```
pi@raspberrypi ~ $ dmesg | grep tty

[ 0.000000] Kernel command line: dma.dmachans=0x7f35 bcm2708_fb.fbwidth=656 bcm2708_fb.fbheight=416 bcm2709.boardrev=0xa01041 bcm2709.serial=0x93f9c7f9 smsc95xx.macaddr=B8:27:EB:F9:C7:F9 bcm2708_fb.fbswap=1 bcm2709.disk_led_gpio=47 bcm2709.disk_led_active_low=0 sdhci-bcm2708.emmc_clock_freq=250000000 vc_mem.mem_base=0x3dc00000 vc_mem.mem_size=0x3f000000 dwc_otg.lpm_enable=0 console=tty1 console=ttyAMA0,115200 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait

[ 0.001774] console [tty1] enabled

[ 0.749509] dev:f1: ttyAMA0 at MMIO 0x3f201000 (irq = 83, base_baud = 0) is a PL011 rev3

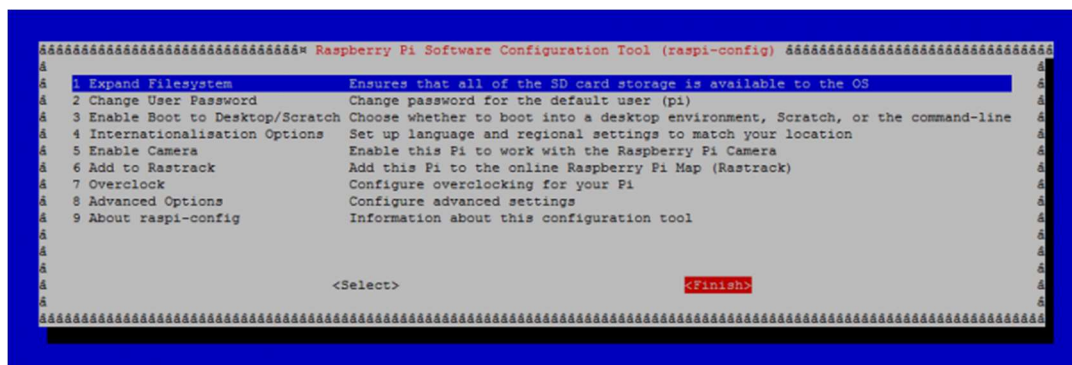
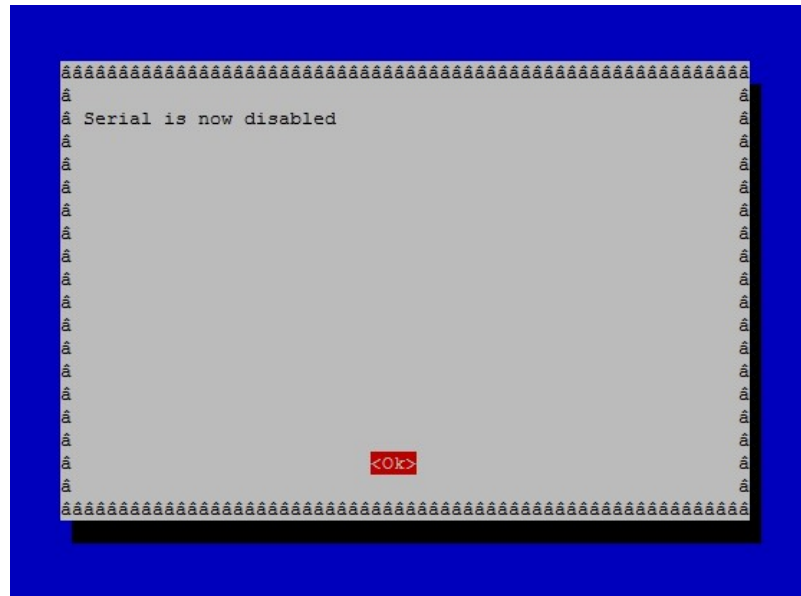
[ 1.268971] console [ttyAMA0] enabled

pi@raspberrypi ~ $
```

Last line indicates that the console is enabled on the serial port ttyAMA0, so we disable it

Run the configuration command and follow the instructions below

```
sudo raspi-config
```

Reboot and try with

```
dmesg | grep tty
```

Output now is

```
pi@raspberrypi ~ $ dmesg | grep tty
```

```
[ 0.000000] Kernel command line: dma.dmachans=0x7f35 bcm2708_fb.fbwidth=656 bcm2708_fb.fbheight=416 bcm2709.boardrev=0xa01041 bcm2709.serial=0x93f9c7f9 smsc95xx.macaddr=B8:27:EB:F9:C7:F9 bcm2708_fb.fbswap=1 bcm2709.disk_led_gpio=47 bcm2709.disk_led_active_low=0 sdhci-bcm2708.emmc_clock_freq=250000000 vc_mem.mem_base=0x3dc00000 vc_mem.mem_size=0x3f000000 dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait
```

```
[ 0.001769] console [tty1] enabled
```

```
[ 0.749438] dev:f1: ttyAMA0 at MMIO 0x3f201000 (irq = 83, base_baud = 0) is a PL011 rev3
```

Now we can use the serial ttyAMA0. We connect an adapter usb / serial, then we will try to establish a communication between the two serial ports; obviously in a practical application to every serial we could connect a device, for example a modem, RFID reader etc.

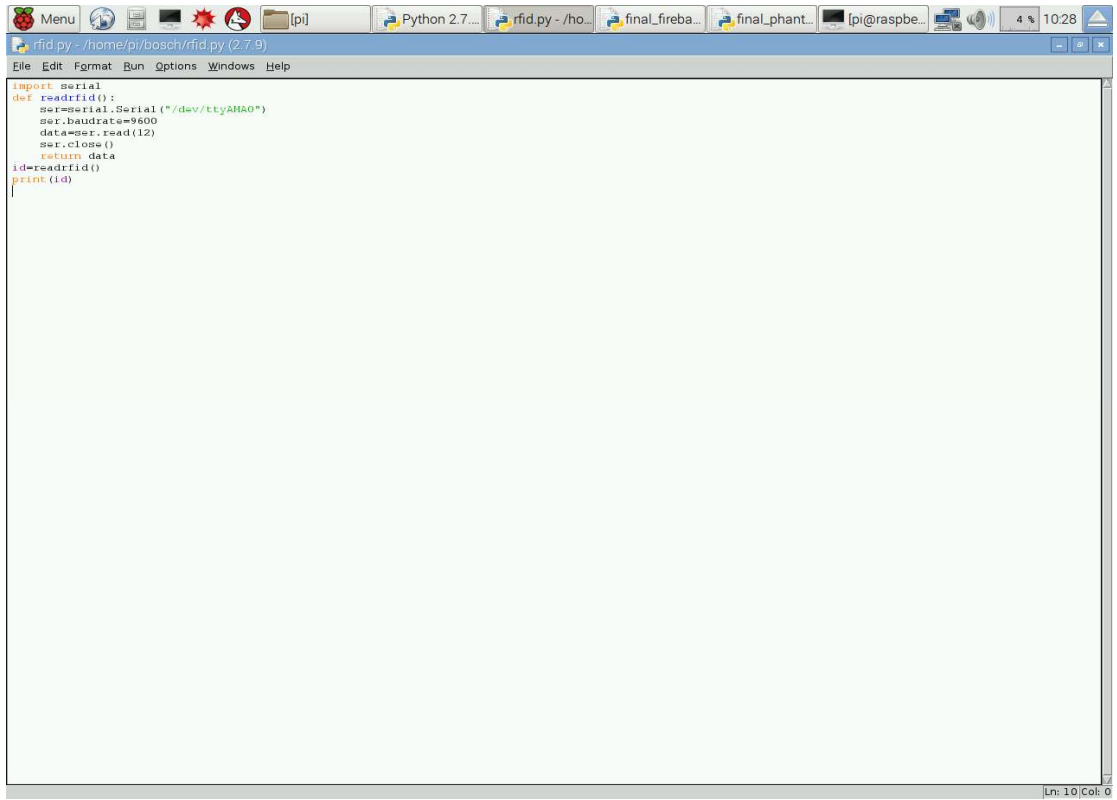
Open the Terminal of RaspBerry Pi and type:

```
pi@raspberrypi ~ $ sudo idle
```

Once the Idle opens, create a new file and type the following codes.

The code:

```
import serial
def readrfid():
    Ser=serial.Serial("/dev/ttyAMA0")
    ser.baudrate=9600
    data=ser.read(12)
    ser.close()
    return data
id=readrfid()
print(id)
```



The image shows a screenshot of a Raspberry Pi desktop environment. The top panel includes a menu icon, system icons (network, volume, battery), and the time 10:28. The taskbar contains several application icons: a terminal window, a file manager window showing the path /home/pi, a Python 2.7 terminal window, and three windows related to an RFID project: 'rfid.py - /ho...', 'final_fireba...', and 'final_phant...'. The active window is a text editor titled 'rfid.py - /home/pi/bosch/rfid.py (2/7/9)'. The editor contains the following Python code:

```
import serial
def readrfid():
    ser=serial.Serial ("/dev/ttyAMA0")
    ser.baudrate=9600
    data=ser.read(12)
    ser.close()
    return data
id=readrfid()
print(id)
```

The status bar at the bottom right of the editor window shows 'Ln: 10 | Col: 0'.

Data Logging using Phant server

If the sensor will detect the vehicle, it will activate the RFID reader, which will read the ID in the tag. The following script will take the values of the reader and the sensor and the put the data in the phant cloud.

```
import RPi.GPIO as GPIO

import time

import http, urllib

import socket

import serial

server= "data.sparkfun.com"

publicKey= "VGX6Jn9A15INybNybHYOJ3Z"

privateKey= "9Yglm8pAwzhA76wypoD1"

fields= ["location","rfid","timestamp"]

try:

    def locat(loc, rfid):

        print("Here we go! Press Ctrl+C to exit")

        print("Sending an update!")

        data={}

        data[fields[0]] = location

        data[fields[1]] = rfid

        data[fields[2]] = time.strftime("%A %B %d, %Y %H:%M:%S")

        params = urllib.urlencode(data)

        headers={}

        headers["Content-Type"] = "application/x-www-form-urlencoded"
```



```
headers["Connection"] = "close"
headers["Content-Length"] = len(params)
headers["Phant-Private-Key"] = privateKey

c = httplib.HTTPConnection(server)
c.request("POST", "/input/" + ".txt", params, headers)
r = c.getresponse()
print r.status, r.reason
time.sleep()

except KeyboardInterrupt:
    GPIO.cleanup()
```

Now open up the terminal, and navigate to the directory where your script lives, and run it like this:

```
pi@raspberrypi ~/code/phant-pi $ sudo python phant-raspi.py
```

With the script running, a handful of messages will appear each time a message is sent. You want to see **“200 OK”** after the **“Sending an Update!”** message. That should indicate that the HTTP POST was successfully sent.

```
Menu [pi] Python 2.7... rfid.py - /ho... final_fireba... final_phant... [pi@raspbe... 10:27  
final_phant.py - /home/pi/bosch/final_phant.py (2.7.9)  
File Edit Format Run Options Windows Help  
import RPi.GPIO as GPIO  
import time  
import urllib, urllib  
import socket  
import serial  
  
server = "data.sparkfun.com"  
publicKey = "%0x%0n%a151HybH0J3Z"  
privateKey = "%9gim%a%h3%9%wpp01"  
fields = ["loc", "rfid", "timestamp"]  
  
try:  
    def locat (loc, rfid):  
        print("Here we go! Press CTRL+C to exit!")  
        print("Sending an update!")  
        data = {}  
        data[fields[0]] = loc  
        data[fields[1]] = rfid  
        data[fields[2]] = time.strftime("%A %B %d, %Y %H:%M:%S %Z")  
        params = urllib.urlencode(data)  
  
        headers = {}  
        headers["Content-Type"] = "application/x-www-form-urlencoded"  
        headers["Connection"] = "close"  
        headers["Content-Length"] = len(params)  
        headers["Phant-Private-Key"] = privateKey  
        |  
        c = urllib.HTTPConnection(server)  
        c.request("POST", "/input/" + publicKey + ".txt", params, headers)  
        r = c.getresponse()  
        print r.status, r.reason  
        time.sleep(1)  
  
except KeyboardInterrupt:  
    GPIO.cleanup()
```

Ln: 27, Col: 0

Connection with the FireBase Server for app to collect the data

FireBase server is used as phant cloud is not accessible for app development. Therefore, we shall put the data in FireBase server, so that the app can access the real-time data from it.

- Register an account in the FireBase account, in order to access it.
- Copy the given URL and paste in the following script.
- Check for network connectivity and run the code.

```
import urllib2
import calendar
import time
import json
url="https://rfidbsp.firebaseio.com/rfid"
def locat(loc, rfid):
    postdata = {
        'time': time.strftime("%A %B %d, %Y %H:%M:%S")
        'location': loc,
        'id': rfid
    }
    req= urllib2.Request(url)
    req.add_header('Content-Type', 'application/json')
    data = json.dumps(postdata)
    response=urllib2.urlopen(req,data)
```

The image shows a terminal window on a Raspberry Pi. The window title is "firebase.py - /home/pi/bosch/firebase.py (2.7.9)". The terminal displays the following Python code:

```
import urllib2
import calendar
import time
import json
url='https://rfidbsp.firebaseio.com/r'
def locat(loc,rfid):
    postdata={
        'time':time.strftime("%A :
        'location':loc,
        'id':rfid
    }
    req=urllib2.Request(url)
    req.add_header('Content-Type','ap
    data=json.dumps(loc)
    response=urllib2.urlopen(req,data)
```

The code defines a function `locat` that takes `loc` and `rfid` as arguments. It constructs a JSON object with fields for `time`, `location`, and `id`. It then uses `urllib2` to send a POST request to the URL `https://rfidbsp.firebaseio.com/r` with the JSON data.

At the bottom right of the terminal window, the status "Ln: 13, Col: 4" is visible.

Interfacing Raspberry Pi with IR sensors for product tracking (The running script)

```
import serial

import time

import Rpi.GPIO as GPIO

from rfid import readrfid

from fire import locat

GPIO.setmode(GPIO.BOARD)

GPIO.setup(11,GPIO.IN)

GPIO.setup(12,GPIO.IN)

GPIO.setup(13,GPIO.IN)

GPIO.setup(15,GPIO.IN)

GPIO.setup(16,GPIO.IN)

GPIO.setup(18,GPIO.IN)

while True:

    loc1= GPIO.input(11)

    loc2= GPIO.input(12)

    plant1= GPIO.input(13)

    plant2= GPIO.input(15)

    customer= GPIO.input(16)

    park= GPIO.input(18)

    if plant1 == True:

        print("Parcel has reached plant 1")
```

```
x="a"  
y=readrfid()  
locat(x,y)  
time.sleep(1)
```

```
elif plant2 == True:
```

```
print("Parcel has reached plant 2")  
x="b"  
y=readrfid()  
locat(x,y)  
time.sleep(1)
```

```
elif loc1 == True:
```

```
print("Parcel has reached location 1")  
x="c"  
y=readrfid()  
locat(x,y)  
time.sleep(1)
```

```
elif loc2 == True:
```

```
print("Parcel has reached location 2")  
x="d"  
y=readrfid()  
locat(x,y)  
time.sleep(1)
```

```
elif customer == True:
```

```
        print("Parcel has reached customer")

        x="e"

        y=readrfid()

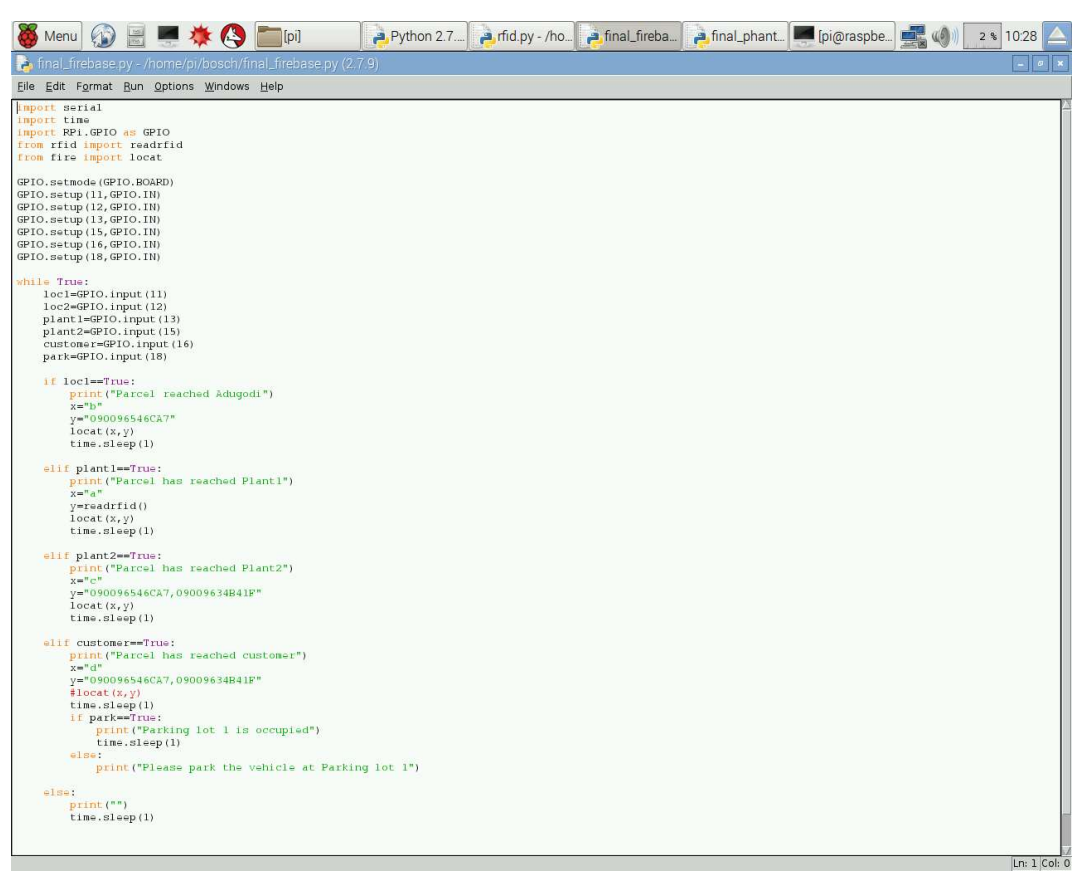
        locat(x,y)

        time.sleep(1)

else:

    print("")

    time.sleep(1)
```



The screenshot shows a terminal window on a Raspberry Pi. The window title is 'final_firebase.py - /home/pi/bosch/final_firebase.py (2.7.9)'. The code in the terminal is as follows:

```
import serial
import time
import RPi.GPIO as GPIO
from rfid import readrfid
from fire import locat

GPIO.setmode(GPIO.BOARD)
GPIO.setup(11,GPIO.IN)
GPIO.setup(12,GPIO.IN)
GPIO.setup(13,GPIO.IN)
GPIO.setup(15,GPIO.IN)
GPIO.setup(16,GPIO.IN)
GPIO.setup(18,GPIO.IN)

while True:
    loc1=GPIO.input(11)
    loc2=GPIO.input(12)
    plant1=GPIO.input(13)
    plant2=GPIO.input(15)
    customer=GPIO.input(16)
    park=GPIO.input(18)

    if loc1==True:
        print("Parcel reached Adugodi")
        x="h"
        y="090096546CA7"
        locat(x,y)
        time.sleep(1)

    elif plant1==True:
        print("Parcel has reached Plant1")
        x="a"
        y=readrfid()
        locat(x,y)
        time.sleep(1)

    elif plant2==True:
        print("Parcel has reached Plant2")
        x="c"
        y="090096546CA7,09009634B41P"
        locat(x,y)
        time.sleep(1)

    elif customer==True:
        print("Parcel has reached customer")
        x="d"
        y="090096546CA7,09009634B41P"
        #locat(x,y)
        time.sleep(1)
        if park==True:
            print("Parking lot 1 is occupied")
            time.sleep(1)
        else:
            print("Please park the vehicle at Parking lot 1")

    else:
        print("")
        time.sleep(1)
```

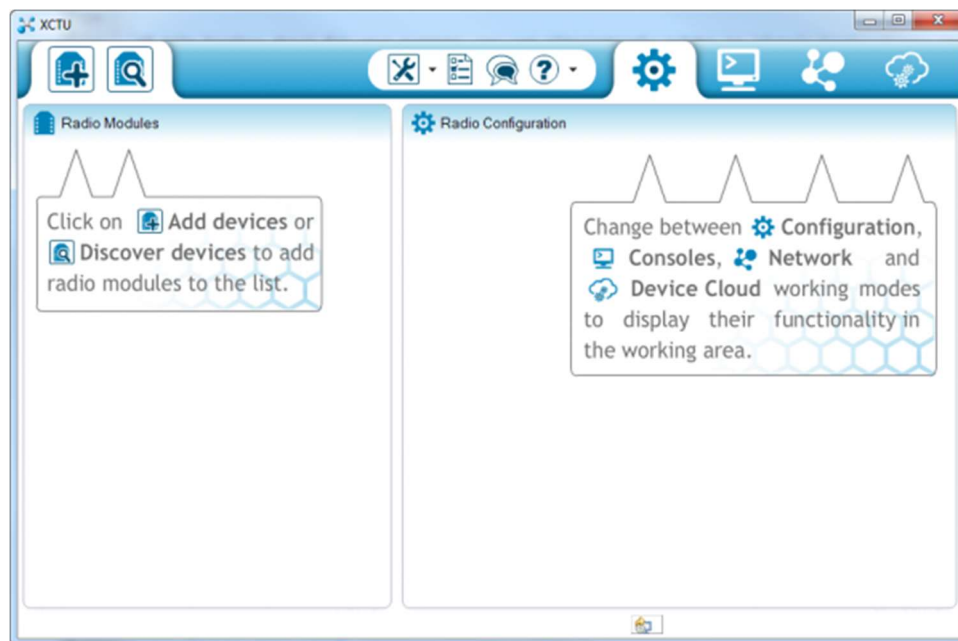
Setting up the XBees

X-CTU is free software, provided by Digi (the manufacturer of XBee), which we use to configure and manage XBees, and test XBee networks.

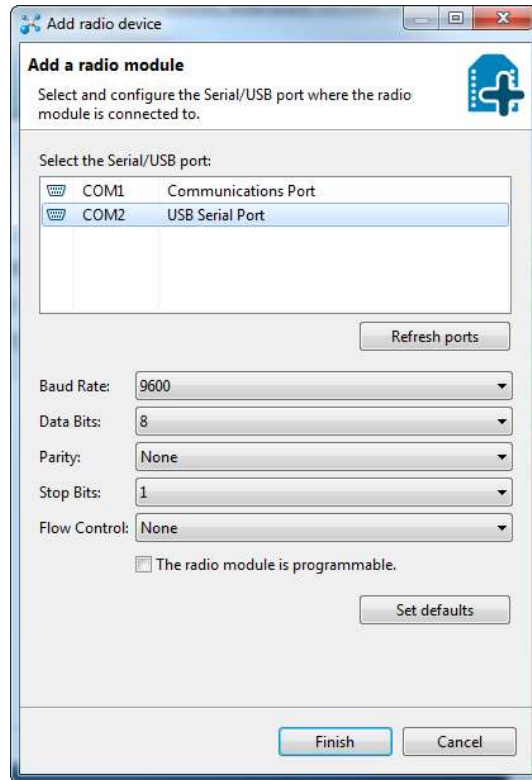
Adding XBees

Before continuing on, make sure you've plugged an XBee (correctly) into your Explorer, and have the Explorer plugged into your computer. When you installed the drivers for your Explorer it should have been assigned port number. You'll need that shortly.

After initially opening X-CTU, you'll be presented with a window like this:



To add XBee(s), click the "Add device" icon in the upper-left part of the window. That will prompt this screen to show up:

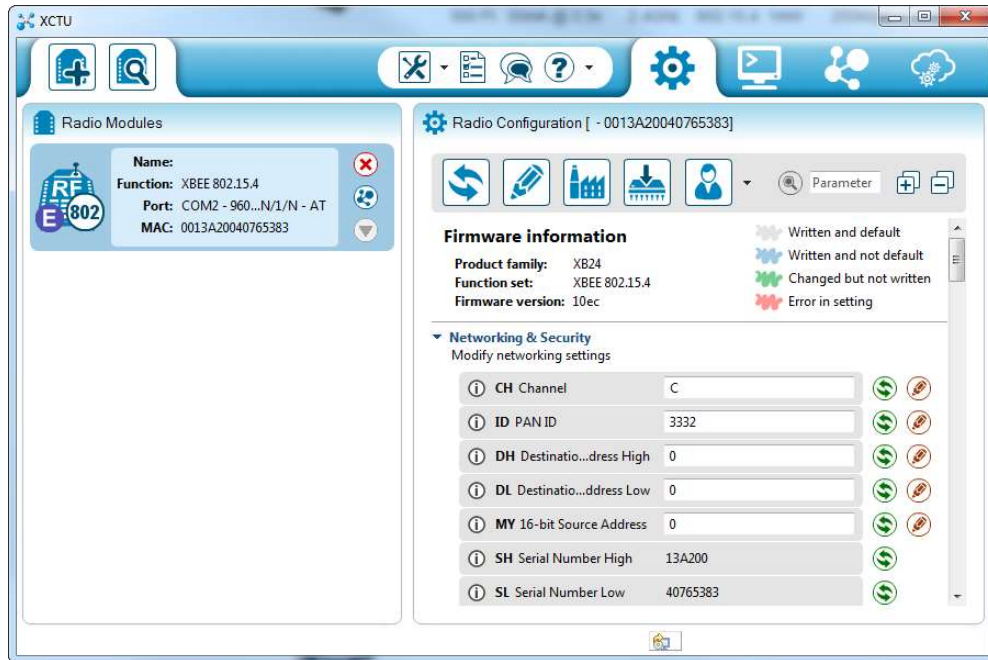


Select communication port. If the Serial Explorer is not showing up, make sure the switch onboard is set to “On”

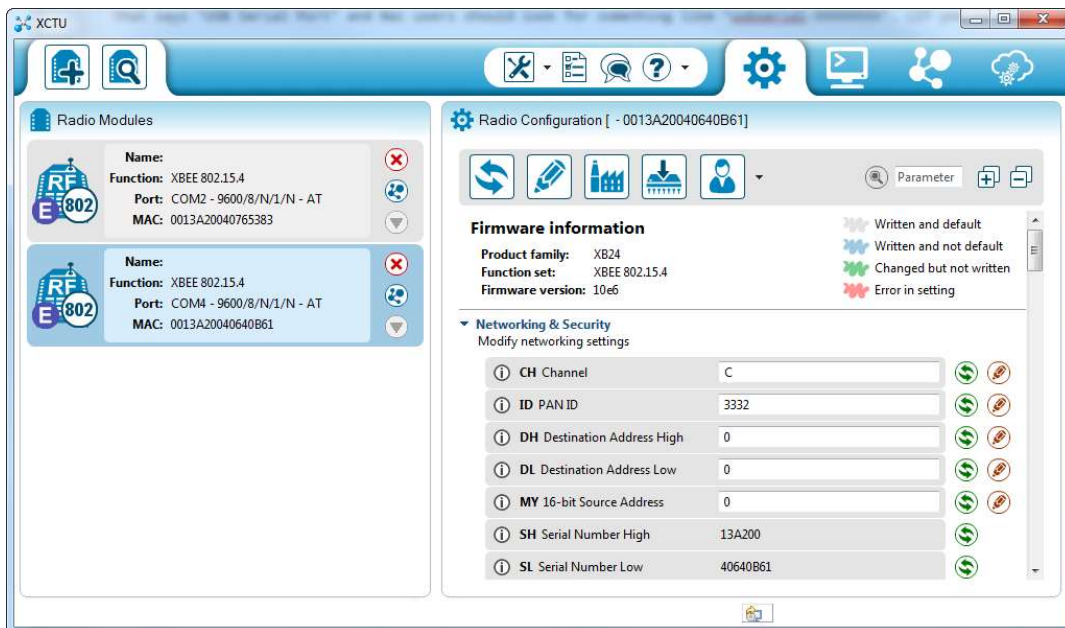
This window also allows you to specify more specific serial characteristics like baud rate, data bits, and stop bits. Click Finish.

A “Discovering radio modules...” window will briefly scroll by, after which you should be presented with the original window, but with an addition to the “Radio Modules” section on the left.

Click that new module, and wait a few seconds as X-CTU reads the configuration settings of XBee.



As you can see by scrolling down the right half, there are a lot of configuration settings available. We'll get to some of those later. For now, though, verify that the configurable settings visible in the screenshot above match those of your XBee (channel=C, PAN ID=3332, DH=0, DL=0, MY=0).



Now, connect one of the XBee to the Raspberry Pi and the other to the IR sensor at parking lot. Type the following code to get the data from the XBee.

```
import serial

ser=serial.Serial('/dev/ttyUSB0', 9600)

while True:
    if ser.readline() == 0x7E:
        for i in range (0,19):
            byte.discard=ser.readline()
        park=ser.readline()
        if park == True:
            print "Parking lot 1 is occupied"
        time.sleep(1)
    else:
        print "Please park the vehicle at Parking lot 1"
```