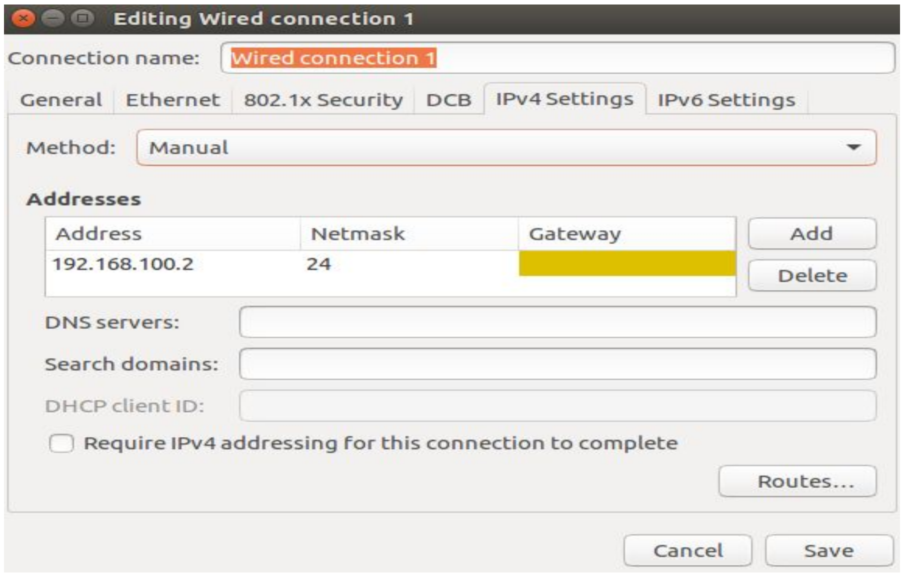
Interfacing Raspberry Pi and PC:

This document describes the steps for interfacing RPi and PC. The aim was to run a single roscore on the PC and run packages on the RPi which can communicate with packages running on the PC. There are three things that need to be done for this : setting static IPs and adding hostnames, adding ROS_MASTER_URI and ROS_IP variables and last how we used screen to execute bash scripts.

Step 1:

First we connected the RPi to PC using a LAN cable and then using GUI we set static IPs on both systems.



Editing Wired connection 1

Connection name: Wired connection 1

General | Ethernet | 802.1x Security | DCB | IPv4 Settings | IPv6 Settings

Method: Manual

Addresses

Address	Netmask	Gateway
192.168.100.2	24	

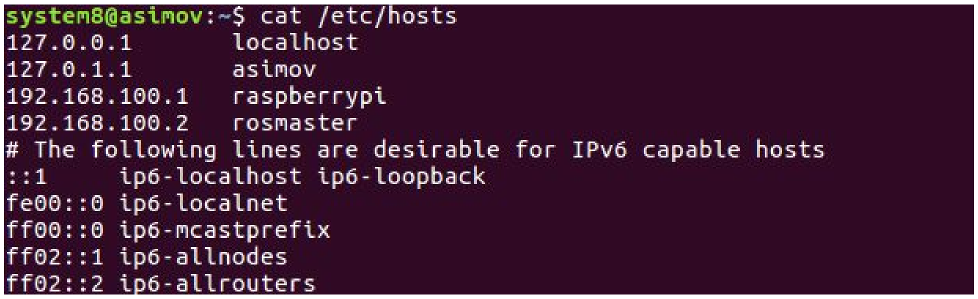
DNS servers:

Search domains:

DHCP client ID:

☐ Require IPv4 addressing for this connection to complete

As you can see from above 192.168.100.2 was set to PC and 192.168.100.1 was set to RPi. The next step is to add the hostnames to the file /etc/hosts on both systems.



```
system8@asimov:~$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       asimov
192.168.100.1   raspberrypi
192.168.100.2   rosmaster
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

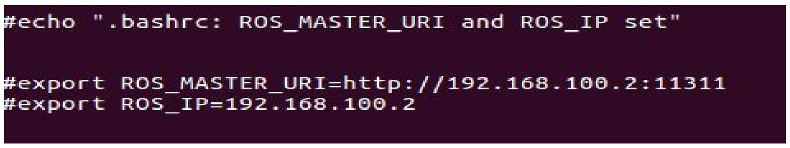
You can see from the above image that domain name of “raspberrypi” was assigned to RPi and “rosmaster” was assigned to the PC. This enables us to ssh into the RPi by typing ssh user_name@raspberrypi instead of user_name@192.168.100.1.

The next step is to setup PublicKey authentication and copy the generated public key TO THE RPi. This will enable us to ssh without having to enter the password every time.

Step 2:

In order to run ROS on multiple machines we have to set a couple of variables. The details are present in the document Multiple ROS computers.pdf.

We added the following line to .bashrc file on both systems so that we can skip having to manually setup the variable everytime.



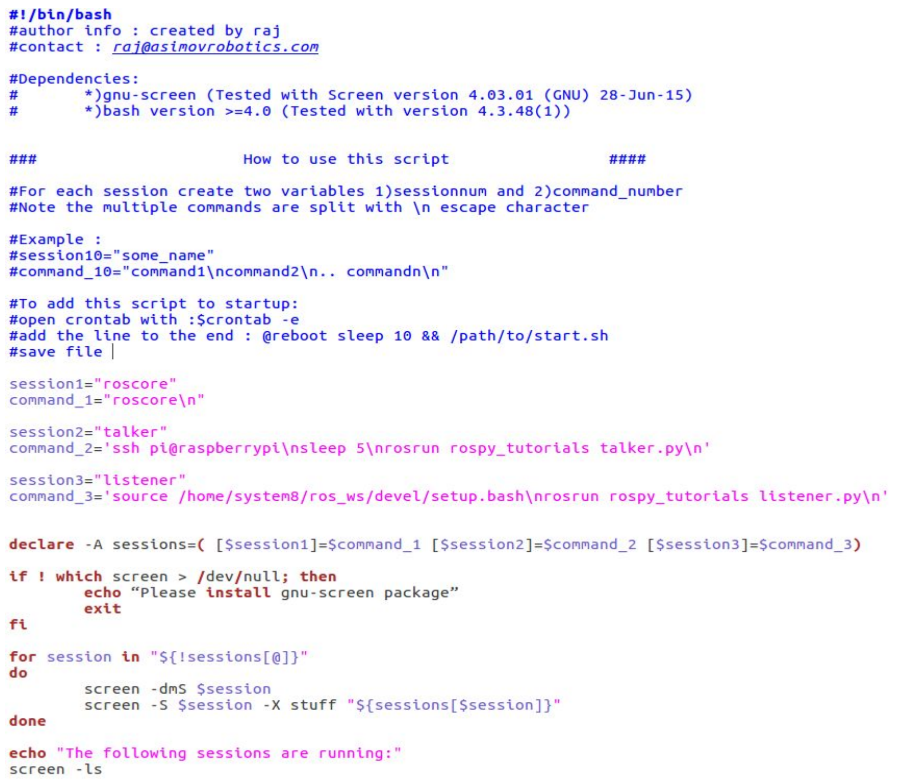
```
#echo ".bashrc: ROS_MASTER_URI and ROS_IP set"

#export ROS_MASTER_URI=http://192.168.100.2:11311
#export ROS_IP=192.168.100.2
```

The above image shows the end of .bashrc file on PC. **Note that on the RPi ROS_IP variable is set to 192.168.100.1 !. Also add “source /path/to/catkin_ws/devel/setup.bash” line to .bashrc on the RPi.**

Once this is done, download ros_tutorials from github and copy the rospy_tutorials package into you catkin workspace on both systems. Then launch roscore and listener.py on the desktop, launch talker.py on RPi. Now the listener should be able to subscribe to the topic published by the talker node on RPi.

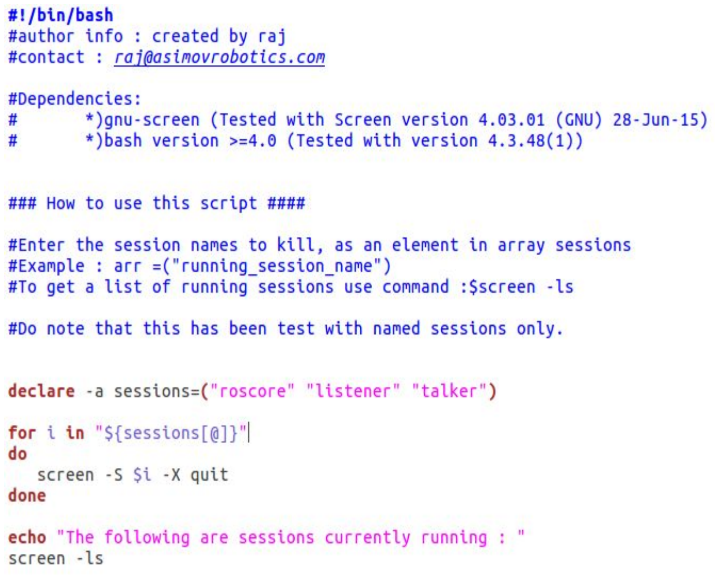
Step 3:

We used gnu-screen to write bash scripts that will automatically launch the roscore and corresponding nodes on both systems. These scripts are present in Linux directory of the drive.



```
#!/bin/bash
#author info : created by raj
#contact : raj@asimovrobotics.com

#Dependencies:
#       *)gnu-screen (Tested with Screen version 4.03.01 (GNU) 28-Jun-15)
#       *)bash version >=4.0 (Tested with version 4.3.48(1))

###                     How to use this script                 ####

#For each session create two variables 1)sessionnum and 2)command_number
#Note the multiple commands are split with \n escape character

#Example :
#session10="some_name"
#command_10="command1\ncommand2\n.. commandn\n"

#To add this script to startup:
#open crontab with :$crontab -e
#add the line to the end : @reboot sleep 10 && /path/to/start.sh
#save file |

session1="roscore"
command_1="roscore\n"

session2="talker"
command_2='ssh pi@raspberrypi\nsleep 5\nrosrun rospy_tutorials talker.py\n'

session3="listener"
command_3='source /home/system8/ros_ws/devel/setup.bash\nrosrun rospy_tutorials listener.py\n'


declare -A sessions=( [$session1]=$command_1 [$session2]=$command_2 [$session3]=$command_3)

if ! which screen > /dev/null; then
        echo “Please install gnu-screen package”
        exit
fi

for session in "${!sessions[@]}"
do
        screen -dmS $session
        screen -S $session -X stuff "${sessions[$session]}"
done

echo "The following sessions are running:"
screen -ls
```

The above is screen shot of start.sh script. This script launches 3 sessions names roscore, listener and talker, then it executes a series of commands in each session.



```
#!/bin/bash
#author info : created by raj
#contact : raj@asimovrobotics.com

#Dependencies:
#       *)gnu-screen (Tested with Screen version 4.03.01 (GNU) 28-Jun-15)
#       *)bash version >=4.0 (Tested with version 4.3.48(1))


### How to use this script ####

#Enter the session names to kill, as an element in array sessions
#Example : arr =("running_session_name")
#To get a list of running sessions use command :$screen -ls

#Do note that this has been test with named sessions only.


declare -a sessions=("roscore" "listener" "talker")

for i in "${sessions[@]}"
do
   screen -S $i -X quit
done

echo "The following are sessions currently running : "
screen -ls
```

The above is screen shot of stop.sh script. It closes all the sessions that we created by the start.sh script. If not closed then the sessions will continue to run in the background. Use “screen -ls” command check if there are any sessions running in the background.

Lastly these scripts can be set to run automatically after booting. Look into start.sh for description of how to achieve this.