FLAG Digital Backend User Manual

Mark Ruzindana, Mitchell Burnett, and Richard Black

July 19, 2019

Contents

1 Introduction

This document provides details on how to run the FLAG backend. It will mostly go through the procedure used to run the system rather than the complexity behind how everything works. In order to run the backend, you will need to clone the repo called flag_gpu on Github to your account on the FLAG HPCs (acquiring account described in next section), and you will also need access to the python directories on the HPCs. If you have an account and have access to the FLAG HPCs, skip section 2 and 3.

2 Acquiring an account

In order to get an account, contact the IT support at the Green Bank observatory (GBO) (Email: helpdesk-gb@nrao.edu). Inform them that you are working with or for one of the appropriate research groups e.g. The BYU radio astronomy systems group. After this, support should give you an account that will give you access to the machines at GBO.

3 Accessing FLAG HPCs

For an extensive procedure on how to access the machines at GBO, go to: https://science.nrao.edu/facilities/gbt/observing/remote-observing-with-the-gbt. This section will describe how to access the FLAG machines specifically. Any additional information will be taken straight from the previously mentioned web page. We will go over the procedure with a Windows machine, but if a user has a Mac or Linux machine, please refer to the link above.

3.1 Virtual Network Connection (VNC) setup

VNC allows remote connections from a client computer to a server, creating a virtual desktop (desktop image) of the server screen on the client computer screen. The user of the client computer can work almost as if he or she were sitting in front of the screen of the remote computer. VNC continuously compresses and transfers screen shots from the server to the client, which makes for a much faster experience than normal X-forwarding.

VNC comes with most Linux distributions and is easily set up. Mac and Windows users should download and install a VNC Viewer.

VNC for Windows is available from TightVNC (www.tightvnc.com) or from RealVNC (www.realvnc.com). Several commercial versions of VNC are available, but the free edition is suitable for remote GBT observations. For purposes of remote GBT observations, only the VNC viewer need be installed on your computer. The VNC server has already been installed on the GBT control room computers and other appropriate machines in Green Bank. You will also need an SSH client. An SSH client allows you to make a secure SSH connection from your work/home machine to the Linux machines in the GBT control room. That is, with SSH client software running on your computer, you can open a terminal window to the

remote Linux computer. For Windows users, PuTTY is a freeware SSH client. It is available from www.chiark.greenend.org.uk/~sgtatham/putty/download.html. Although other SSH client software exists (e.g. SSH Secure Shell, Secure CRT), our instructions assume you are using PuTTY.
Thus, remote Window users should:

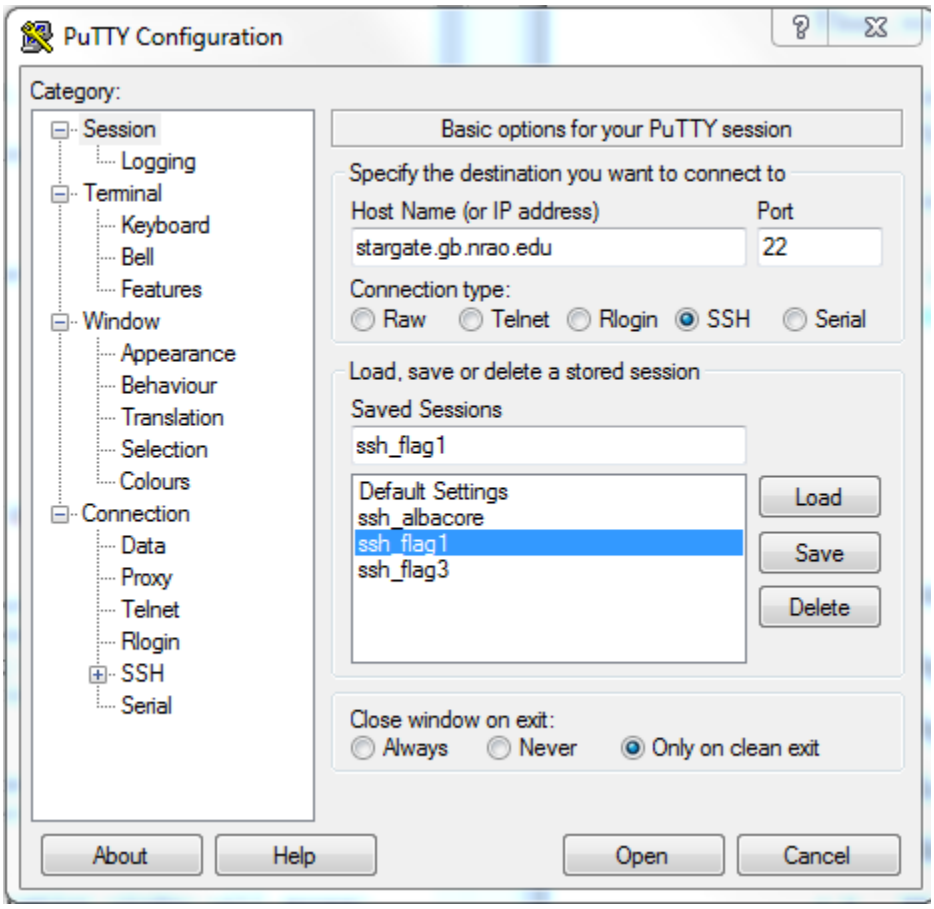
- Download and install VNC unless it is already installed. You need only install the VNC viewer.
- Download and install PuTTY unless it is already installed on your machine.

Mac users should refer to the VNC setup section in https://science.nrao.edu/facilities/gbt/observing/remote-observing-with-the-gbt.
If this is your first session ever with VNC, you need to setup a VNC server password and specify which window manager you would like to use.

Choose a VNC password that is different from your NRAO Linux account password as you may later wish to share your VNC password with others who can then observe your VNC session. For example, if you are having difficulties during a remote observing session, you might wish to share your VNC password with a Green Bank staff member who can access your virtual desktop and suggest solutions to your problems.

To create a VNC server password, you must log into an NRAO computer. The first steps for Linux and Mac OS X users are different from those for Windows users:

- Start up PuTTY on your Windows machine. A PuTTY Configuration window will appear.
- In the configuration window, specify the host name, stargate.gb.nrao.edu and click on Open to obtain a terminal window to the host. After specifying the host name, one can choose Save to save the session for future use. If the host name already appears among the Saved Sessions, double click on the host name to open a terminal window to that host.



- In the PuTTY terminal window to stargate.gb.nrao.edu, log in to your NRAO Linux account if you are not using an SSH agent.

Once you are logged into the NRAO computer, and regardless of your remote computer:

- In the terminal window on the remote (NRAO) computer, type at the Linux prompt:

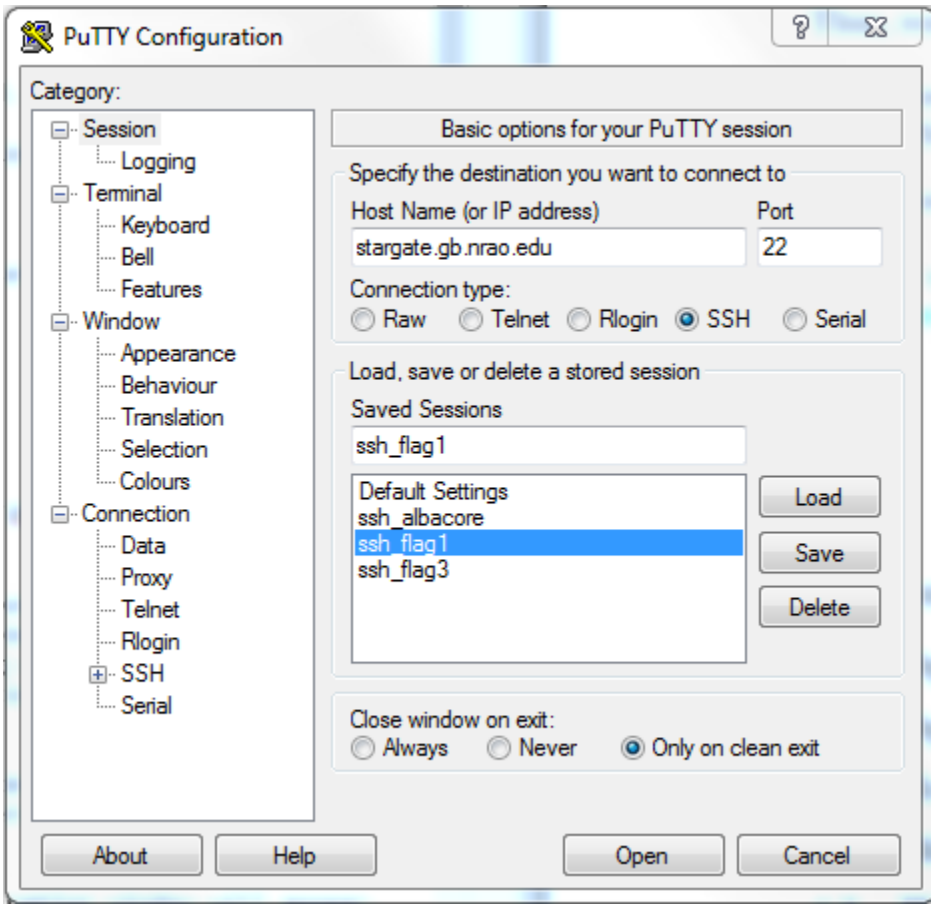
```
vncpasswd
```

- Enter your chosen VNC server password. Remember, this password should be different from your NRAO Linux account password as it will be shared with support staff when you require assistance.

3.2 Setting up a Window Manager

For Windows users, log into an NRAO computer, if you aren't already logged in, using PuTTy:

- Start up PuTTY on your Windows machine. A PuTTY Configuration window will appear.
- In the configuration window, specify the host name, stargate.gb.nrao.edu and click on Open to obtain a terminal window to the host. (After specifying the host name, one can choose Save to save the session for future use. If the host name already appears among the Saved Sessions, double click on the host name to open a terminal window to that host.)



- In the PuTTY terminal window to stargate.gb.nrao.edu, log in to your NRAO Linux account if you are not using an SSH agent.

Once you are logged into the NRAO computer, and regardless of your remote computer:

- In the terminal window on the remote (NRAO) computer, type at the Linux prompt:

```
mkdir ~/.vnc
```

If you have already used VNC in the past, this step is not necessary and you will get a message stating that the 'file' already exists. This is normal.

- If you want to use the *kde*window manager, at the Linux prompt, type:

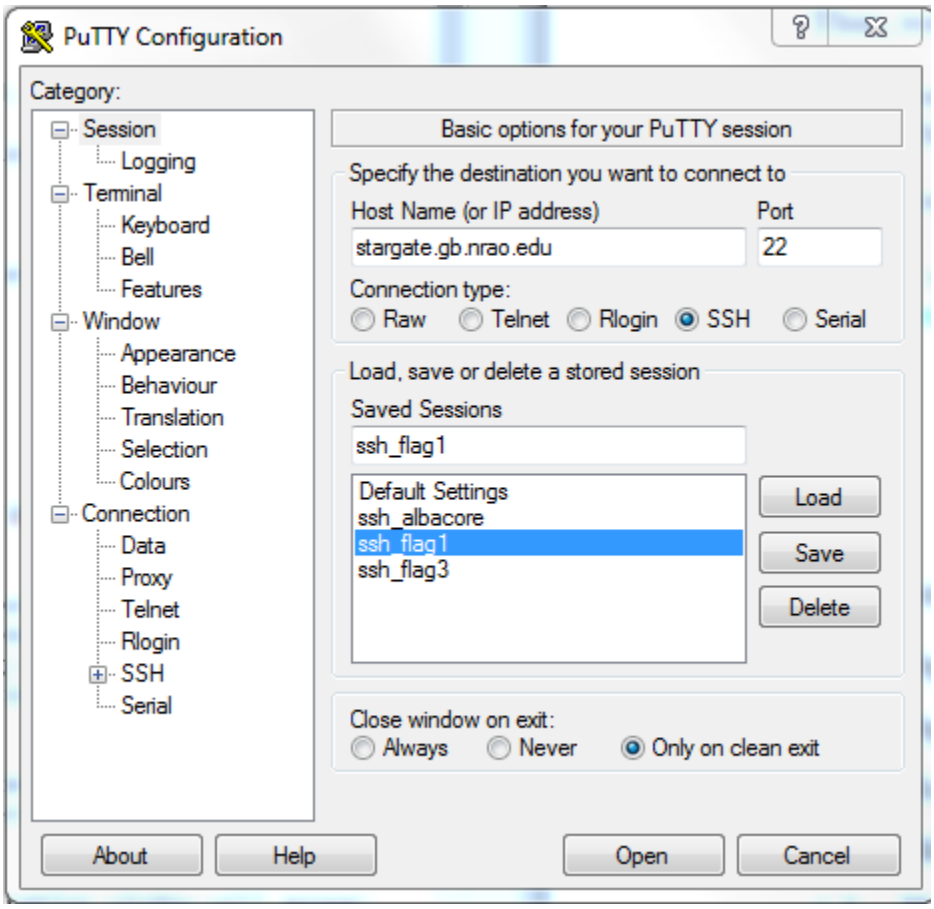
```
cp /users/cclark/xstartup-KDE ~/.vnc/xstartup
```

3.3 VNC session instructions

Each time you observe remotely with the GBT, you will need to follow these instructions.
Note: Please do not start more than one VNC session at a time.
There is only a limited total number of sessions and ports available at any time. Likewise, don't forget to end your VNC server session when you have finished observing.

To set up a session at the remote machine, you must first log into an NRAO computer. The first steps for Linux and Mac OS X users is different than those for Windows users: For Windows users, log into an NRAO computer, if you aren't already logged in, using PuTTy:

- Start up PuTTY on your Windows machine. A PuTTY Configuration window will appear.
- In the configuration window, specify the host name, stargate.gb.nrao.edu and click on Open to obtain a terminal window to the host. (After specifying the host name, one can choose Save to save the session for future use. If the host name already appears among the Saved Sessions, double click on the host name to open a terminal window to that host.)



- In the PuTTY terminal window to stargate.gb.nrao.edu, log in to your NRAO Linux account.

Once you are logged into the NRAO computer, and regardless of your remote computer (For an example we will use flag1, but the same process applies to flag2,3,4 and 5):

- In the terminal window on the remote (NRAO) computer, type at the Linux prompt:

  ```
  ssh flag1.gb.nrao.edu
  ```

  If you are not using an SSH agent, you will be asked to enter your NRAO Linux account username and password.

- At the Linux prompt on *flag1* type:

  ```
  vncserver
  ```

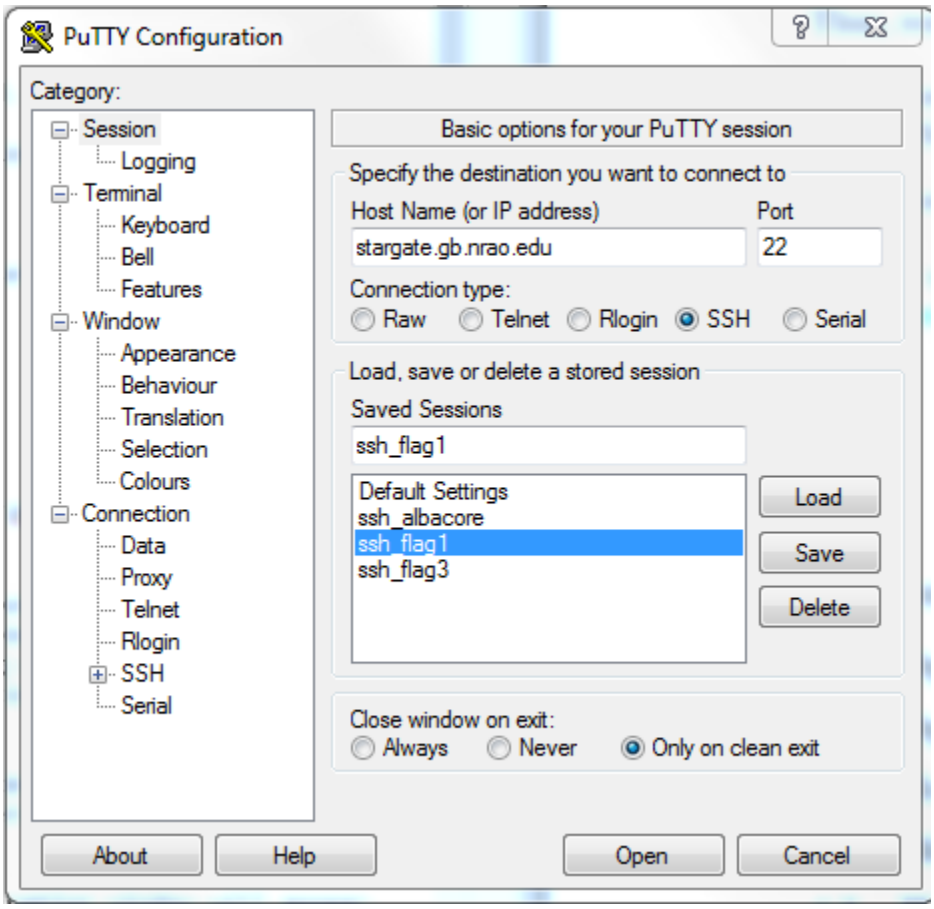
- You can also run vncserverwith color depth or window size options. For example:

```
vncserver [ -geometry 2048x1024 ] [ -depth 8 ] [-pixelformat BGR233]
```

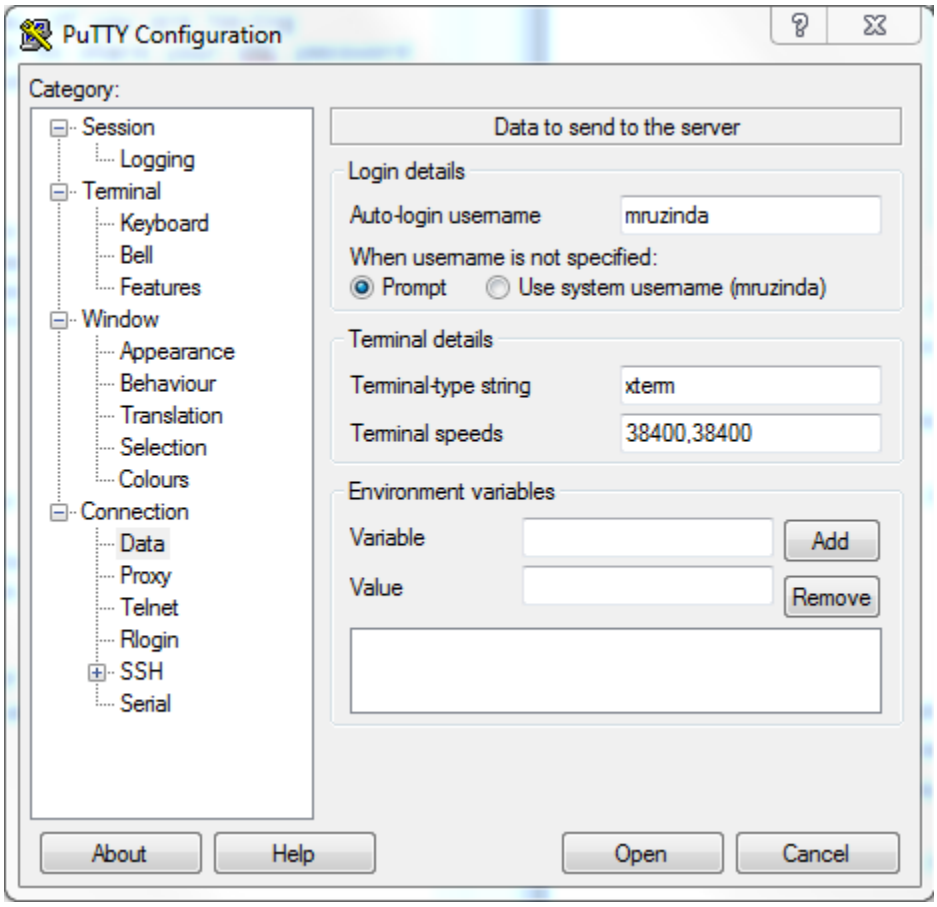
- These options allow you to specify an alternate screen geometry or color depth. The first number for the geometry is the screen width, which we advise should be 2x wider than the size of your local monitor. Choose a screen height that is 1024 or larger, so as to accommodate the typical observing applications. If you have a monitor that is more than 1024 pixels high, choose a screen height that is slightly smaller than the native height of your local monitor.
- Choose 8-bit color to speed up the transfer of displays in the VNC session. Sometimes, it isn't necessary to start vncserver with the -depth option, since your viewer may assume defaults that might actually work better without specifying a color depth. Note that you will have to specify -pixelformat BGR233 along with -depth 8 for astrid to run properly.
- If this is your first session in Green Bank, you will be asked which password should protect your future sessions. As noted above, choose a VNC password that is different from your Linux account password as you may later wish to share this password with others who can then observe your VNC session. For example, if you are having difficulties during a remote observing session, you might wish to share your VNC password with a Green Bank staff member who can access your virtual desktop and suggest solutions to your problems.

You will need to establish an SSH tunnel and start a VNC viewer in order to view the session on your local machine (work/home). The process is different for each computer platform, and we will go over the process for windows users. If you are using another platform, please refer to the link at the beginning of the section. The only difference will be using a FLAG machine vs. *titania*. The windows process is as follows:

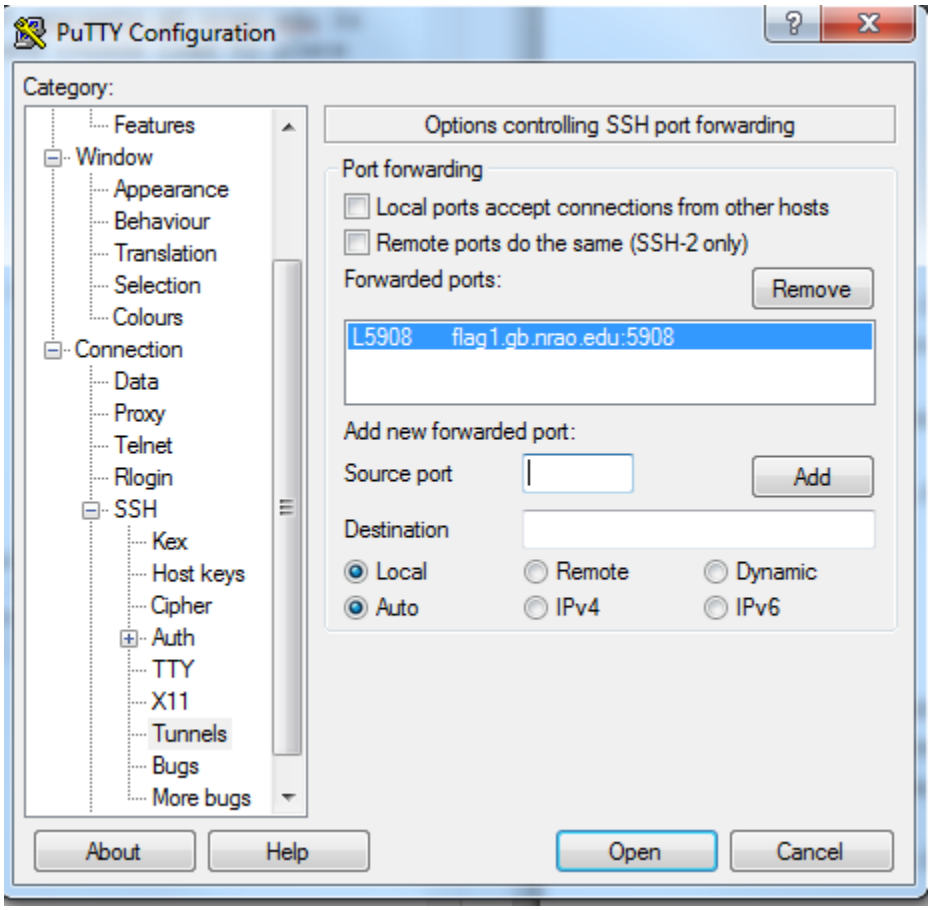
- To establish an SSH tunnel:
  - Start another instance of PuTTY and bring up another PuTTY Configuration window.
  - In this second PuTTYwindow, enter the following information under each Category (listed in the left panel of the window). This information can be saved for future use.
    - Session - Host Name is stargate.gb.nrao.edu. (If stargate.gb.nrao.edu is already listed among the Saved Sessions, click on it and choose Load to place that name in the Host Name section of the window.)



* Connection - Data - Enter your Green Bank Linux account login name as the Auto-loginusername.



* Connection - SSH - Select Enable compression.
* Connection - SSH - Tunnels - Remove any previously used ports with the Remove button. For Source port enter 590n, where n is the VNC session number reported in the first PuTTY window (the VNC server). For Destination, enter flag1.gb.nrao.edu:590n Then choose Add, then Open.



  - ∗ A terminal screen will open to stargate.gb.nrao.edu. If you are not using an SSH agent, you will be prompted for your NRAO Linux account password.
  - ∗ You need type nothing else in this window except exit at the end of the VNC session. The existence of this window serves only to provide the tunnel from your Windows machine to the Green Bank system.

- To start the VNC viewer:
  - Start the VNC viewer on your Windows machine. If using TightVNC, please select the viewer having the "best compression."
  - A popup window will appear, VNC Viewer: Connection Details.
  - Click on Options. Search for and select the option for sharing the connection and then click OK.
  - For Server enter localhost:n, where n is the VNC session number, as before.
  - Next a VNC Viewer: Authentication window will pop up. Enter your vncserver-password (not your NRAO Linux password).

- The VNC Viewer window to *flag1* will now appear on the screen of your Windows machine. In this window you can start astrid and cleo, open xterm, etc, almost as if you were sitting in front of a *flag1* screen.

3.4 Closing a VNC session

If you stop using your VNC viewer but don't kill the vncserver, your session stays alive. If you run your VNC viewer a few hours later or from a different computer, you can continue where you left off. However, there is a limited number of sessions and ports available at any one time. Unless you know you will be using the session again within a few hours, please do not leave vncserver running. If you do, the GBT operator is likely to kill your vncserver session within 24 hours.

Also, please do not start more than one VNC server at a time.

To stop your VNC session, first close your VNC viewer, then kill your VNC server. You'll need a terminal window that is logged into an NRAO computer. If you aren't already logged into an NRAO computer, login using PuTTy:

- Start up PuTTY on your Windows machine. A PuTTY Configuration window will appear.
- In the configuration window, specify the host name, stargate.gb.nrao.edu and click on Open to obtain a terminal window to the host. After specifying the host name, one can choose Save to save the session for future use. If the host name already appears among the Saved Sessions, double click on the host name to open a terminal window to that host.
- In the PuTTY terminal window to stargate.gb.nrao.edu, log in to your NRAO Linux account.

Once you are logged into the NRAO computer:

- In the terminal window on the remote (NRAO) computer, at the Linux prompt type:

```
ssh flag1.gb.nrao.edu
```

- Type:

```
vncserver -kill :n
```

  where n is the VNC session number.

- Disconnect from flag1.gb.nrao.edu by typing exit.
- Disconnect from stargate.gb.nrao.edu by typing exit.

3.5 Troubleshooting

There have been times when a local port is taken by another user or another VNC session. In these rare cases, the recommended port forwarding won't work. To determine if a port is used, the terminal command,

```
netstat -a | grep :59
```

will list all used ports (they may be a delay of a few seconds before this list appears).

In these cases, the tunnel has to be changed to 590m:stargate.gb.nrao.edu:590n where 590m is some unused port numbered somewhere above 5900. And wherever localhost:n occurs in the above instructions, substitute localhost:m.

If you would like to check if you are already running a VNC server (really, there should only be one), use any terminal that is logged into an NRAO computer and type:

```
ls ~YOURLOGIN/.vnc | grep .pid
```

4 Running system during observation

In order to run the system, the user should have a basic understanding of operational modes, dealer/player, and bit/byte/word lock. With this understanding, the user can run the system and acquire/process data accordingly. After going over the details in section 4, the next section provides a good overview of how to run the system. Please go over both section 5 and 6 because these are essential to full operation.

4.1 Operational modes

There are 4 operational modes that are of importance. 3 of them are functional and 1 is still in development. The operational modes are; the coarse channel correlator, fine channel correlator, the real-time beamformer, and the concurrent mode. The coarse channel correlator mode generates covariance matrices for 500 frequency bins (303 kHz wide).

The fine channel correlator has the same structure as the coarse correlator mode except that it includes a fine PFB module for further channelization prior to the correlator. This mode channelizes five coarse frequency channels (303 kHz each, totaling 1.51 MHz) into 160 finer channels (9.5 kHz each, totaling 1.51 MHz). The real-time beamforming mode uses weights developed in post-processing (formed from the coarse channel correlations) and beamforms the data.

4.2 Operating dealer/player

The Dealer/Player system is a Python-based code suite that manages ROACH boards and HPC processes and provides mechanisms to issue user commands and control shared memory. Detailed documentation for the Dealer/Player system can be found at [1]. In summary, a "Dealer" acts as a server, issuing commands to its "Players" or clients via zero-MQ socket protocols.

Player Interface.pdf

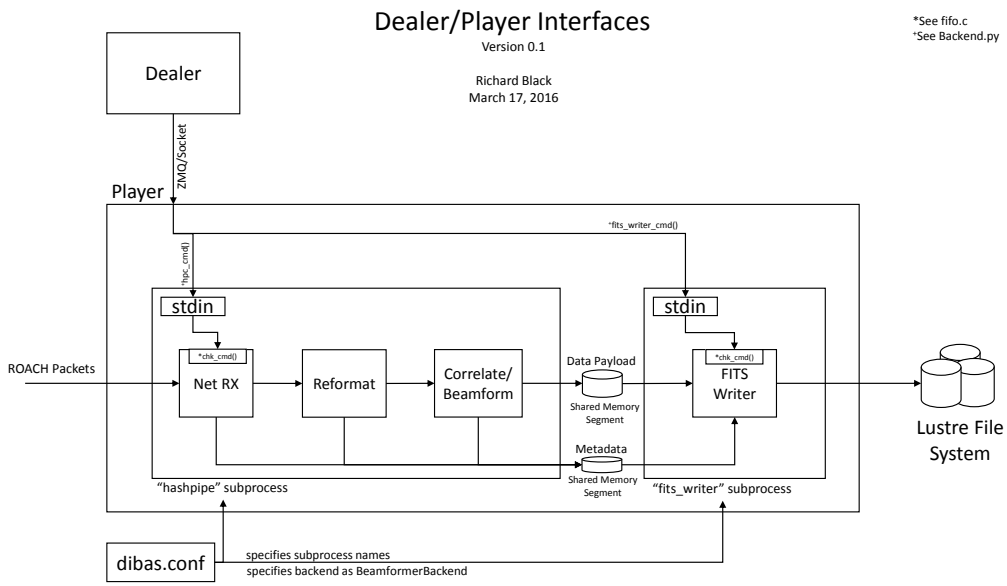


Figure 1: A single dealer/player interface.

The FLAG back end system has a single Dealer that communicates with 20 Players that each manage a single instance of our HASHPIPE code and FITS writer. A single Player is shown in Figure 1. Here the Dealer issues commands to the Player, which in turn translates that command into messages that are then fed into the `stdin` of the HASHPIPE and FITS writer processes.

4.3 Bit/byte lock (provided by Luke Hawkins)

In order to do bit/byte lock, you will need the PAF monitor and control GUI. Talk to Luke Hawkins and IT support in Green Bank to gain access to this GUI. You will also need to get an account on one of the machines at Green Bank called albacore. This will be where you use the PAF monitor and control GUI. The following 2 sections show the procedure of manual bit/byte lock and automated bit/byte lock.

4.3.1 Manual bit/byte lock

1. Open from [E:/For Mark/Testing/PAF - Monitor and Control (application)
2. Run Labview (white arrow, top left). Before running, deselect any ROACHs that you don't want to communicate with ("Enable ROACH COM", Config tab).
3. Turn on the netburner & load attenuation settings (under netburner tab, select "Power

On", wait until it is deselected, then click "LOAD" at for the attenuator values - these should be approximately 9, you can put "9" in the "Array Set Value" field, then "Populate Array", then "LOAD" if you want to make sure the arrays are being written to correctly.

4. Under the Config tab, click "Noise Source" under the "RF Switch" field (bottom, center-left).

5. Under "Balancing/Align" tab, click "Align Bit". When done, you should see green dots (dots are not expected to be green for A8 & E8).

6. Under "Calibration Files", select the 'choose path' icon for the "Old Style Folder Path" field, and select E:/For Mark/ftp.cv.nrao.edu/NRAO-staff/jcastro/PAF. Then click "Populate Cal Array" for that same field. Then select the correct LO freq under "Sideband Cal LO Freq (MHz" fild, and hit "OK". Green lights should light up under "Load Cal Success Blade". When selecting for the LO freq, feel free to choose whatever available freq point is closest to what you're using. The coefficients don't change too much with LO Frequency.

7. Under the "Config" tab, under the "RF Switch", click "Test Tone". You can then set the desired power level and frequency under the "Test Tone Controls" field - then click "Set Test Tone" to implement your changes (typically, I've been setting Test Tone Power to 10dBm to see the tone well for the next step).

8. Now, byte alignment. First, go to the "Balancing/Align" tab, set the "Image Rejection Threshold" field to 10, and click "Align Byte". Currently, this method hasn't had 100"SBS Data" tab, and using the "Channel Controls" on the left, swap bytes on any channels where there is a tone on both sides of the center of the bandpass. For this step, it helps if the Test Tone is 30-50MHz offset from the LO.

9. Now, if you want to use the dipoles to observe, click "OFF" on the "RF Switch" field under the "Config" tab.

General - Keep an eye on the blade temps - I think they shut down at 30 degrees - if they get too warm, turn the blades off for awhile ("Power OFF", under Netburner tab), or ask me to put more ice in the bucket. Don't forget to turn the blades off when you're done for the night. Also, use the "STOP" button (red letters) to stop the Labview, then close the Labview.

4.3.2 Automated bit/byte lock

Here's the link to the PAF health-monitor site: http://paf2015.gb.nrao.edu/INDEX.HTM

Here's the instructions (assuming starting from nothing):

1. Reset the netburner with the 'reboot_paf' batch file (must be done from windows computer). Change the last 3 digits of the IP to 208 and click update.

2. From Ipython, enter the following (to bring in the libraries, etc...) under /home/groups/flag/dibas/lib/python/f_engine_config where the configuration scripts (f_engine_config_lo....py) are found or refer to the path when running the script:

```
import corr, time, struct, sys, logging, socket, os
from numpy import genfromtxt
import numpy as np
import nb_util as nb

nb.connect()
nb.set_rf_switch('NS')
```

3. From Ipython enter the following (to turn on blades and set attenuation):

```
nb.send_recv('PS0')
nb.set_all_blades('9')
```

4. Turn on GUI (only using white arrow in top left corner) for real-time analysis of time domain data and SBS data.

5. From ipython enter the following (to set bit-lock - wait for outputs from the scripts to finish):

```
os.system("python f_engine_config.py flagr2-1  -n -f 0  -b 1 -i
0xaaaaaaaa")
os.system("python f_engine_config.py flagr2-2  -n -f 1  -b 1 -i
0xaaaaaaaa")
os.system("python f_engine_config.py flagr2-3  -n -f 2  -b 1 -i
0xaaaaaaaa")
os.system("python f_engine_config.py flagr2-4  -n -f 3  -b 1 -i
0xaaaaaaaa")
os.system("python f_engine_config.py flagr2-5  -n -f 4  -b 1 -i
0xaaaaaaaa")
```

6. The script expects TT +950MHz off of LO. From Ipython enter the following (to set RF switch to TT):

```
nb.set_rf_switch('TT')
```

7. Byte Lock (also loads coefficients for 1450MHz LO):

   (a) Run scripts:

```
"python f_engine_config.py flagr2-1 -n -f 0 -c 1450 -B 2"
"python f_engine_config.py flagr2-2 -n -f 1 -c 1450 -B 2"
"python f_engine_config.py flagr2-3 -n -f 2 -c 1450 -B 2"
"python f_engine_config.py flagr2-4 -n -f 3 -c 1450 -B 2"
"python f_engine_config.py flagr2-5 -n -f 4 -c 1450 -B 2"
```

   (b) This will output two plots (both pre-alignment). The first is the full bandwidth (upper and lower sideband), the second is the difference (upper-lower) sideband. I could change this to print the before and after byte-alignment full bandwidth plots if you want.

   (c) Using the +950MHz TT offset, a byte-aligned channel should have a nice filter shape in the upper sideband, and a smooth lower sideband.

   (d) Bytes should be aligned now.

8. From ipython enter the following (to turn RF switch off):

```
nb.set_rf_switch('OFF')
```

9. In ipython enter the following (to turn blades off and disconnect the netburner from python):

```
nb.send_recv('PS1')
nb.NB.close()
```

4.4 Word lock

After bit and byte lock, samples need to be aligned and this is called word lock. This is done with 2 files in MATLAB; `snoop_lock.m` and `compare_lock.m` in /home/groups/flag/scripts/matFlag/word_lock. Follow the procedure below to word lock:

- Use `snoop_lock.m` to word lock by running a 1 second scan and typing the generated file name into the ".m" file and record the reference element.
- Then, change the time stamp in the file, `set_delays.py` in the python directory (/home/groups/flag/dibas/lib/python) and run it.
- Run another 1 second scan, type the file name into the file, `compare_lock.m`, and type the current reference element (a variable located at the bottom of the script). This final procedure will verify word lock. Word lock is verified when the values printed in the command line are mostly 0 meaning there is 0 sample offset between elements.

Now you can run dealer/player with the relevant operational mode. If the result at the end isn't mostly zeros, then redo word lock or a quick check is to make sure that the set_delays.py file was run with the correct time stamp.

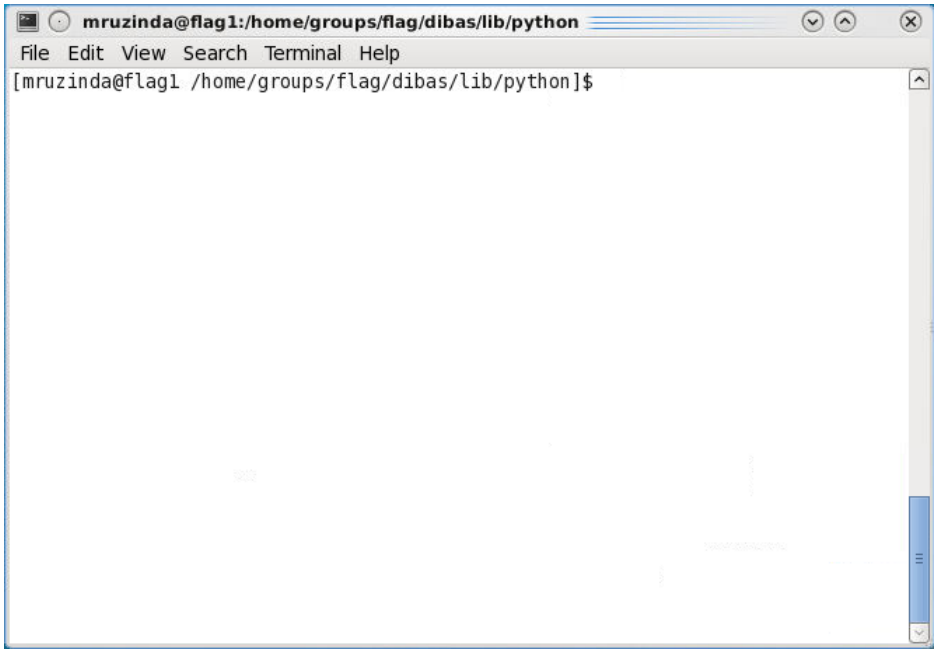
4.5 System operation

This section will provide details on system operation. It will also provide solutions to errors that might be encountered. The steps will be presented with the coarse correlator mode and additional/modified steps for the other operational modes will be provided.

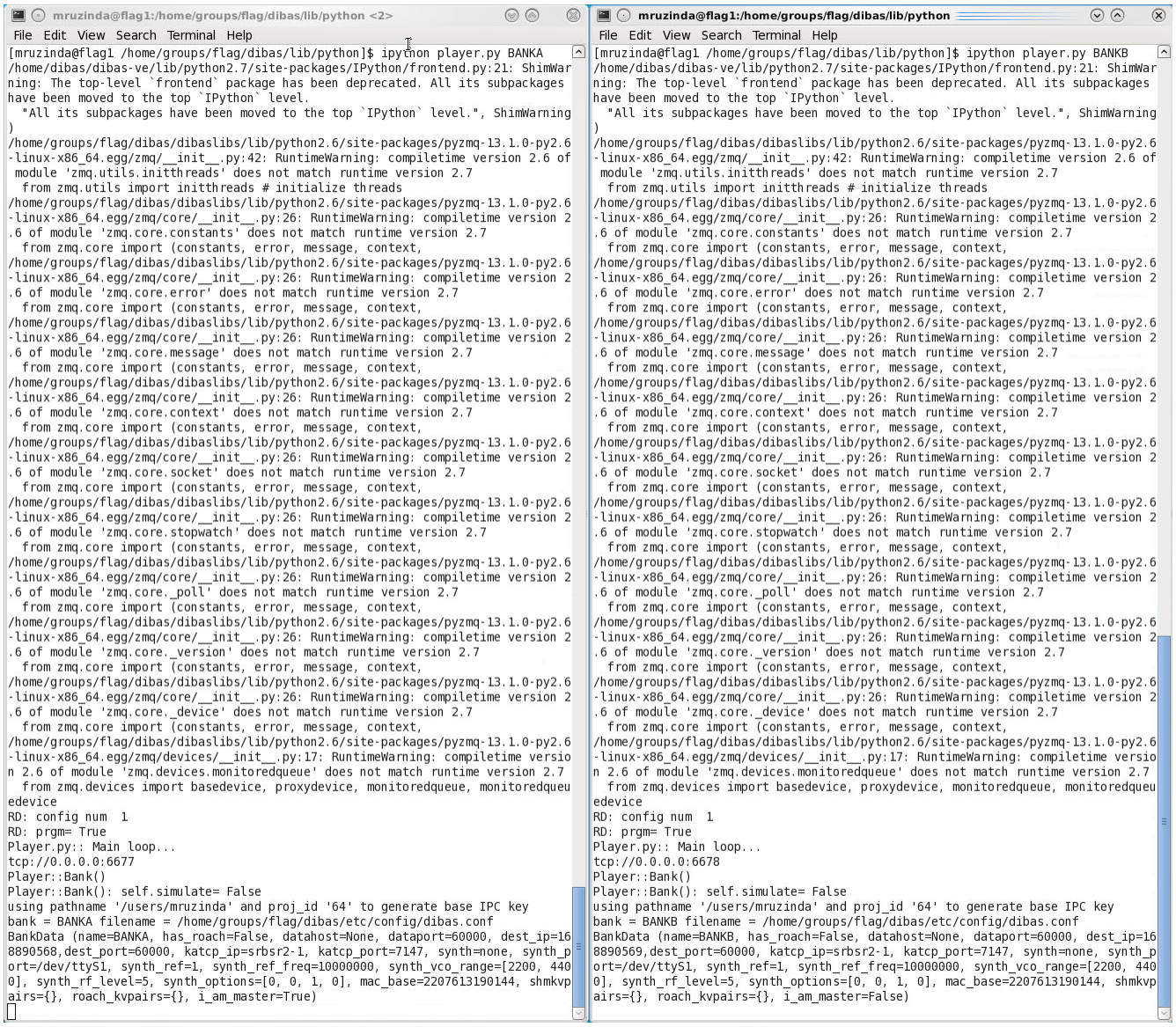
On a side note, to simplify/speed up the typing in commands, you can get the correct commands in ipython (after running them once) via a minimum match and then pressing the up arrow.

In order to run the system, follow these steps shown with 1 or 2 terminals rather than the 20 that should be opened:

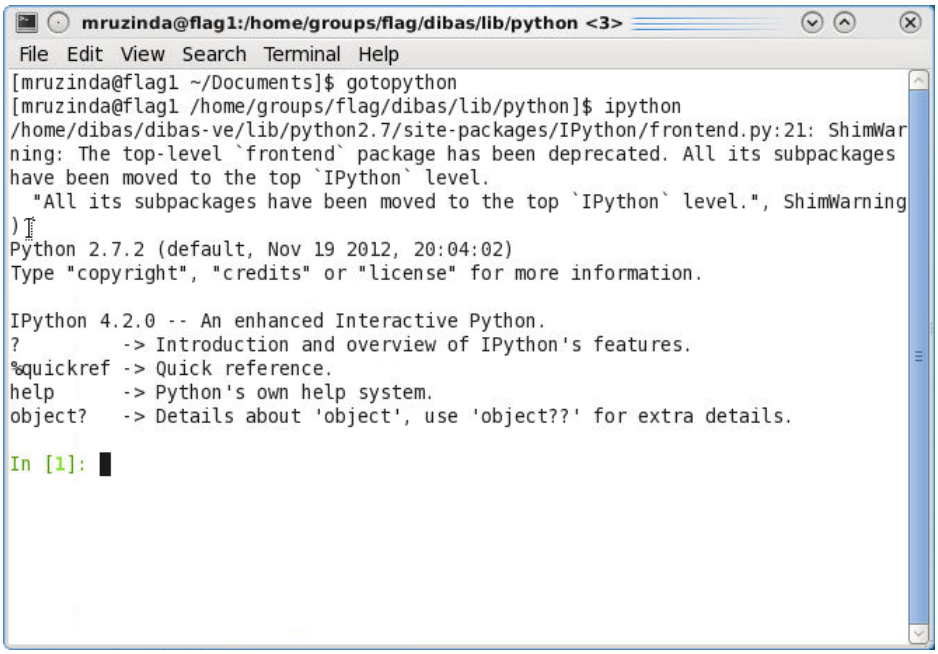
- Go to the python directory on the HPCs.



- Start the players on the appropriate HPCs. The following line is to be entered in each linux terminal (20 terminals, 4 on each HPC):
  - `ipython player.py "bank name in capital letters e.g.  BANKA"`



- Open another terminal for the dealer (this can be on any HPC) open ipython and in the python directory and enter the following lines to set the mode and other parameters:



- `import dealer`
- `d = dealer.Dealer()`
- `d.add_active_player(*d.list_available_players())`
  - Or `d.add_active_players('BANKA', 'BANKB')` if you're trying to start particular players.
- `d.set_mode('FLAG_CALCORR_MODE')`
  - Or `d.set_mode('FLAG_RTBF_MODE')`, `d.set_mode('FLAG_PFBCORR_MODE')` if you're trying to set the real-time beamformer or fine channel correlator modes.
- `d.set_param(int_length = 0.5)` which is typically 0.5 unless performing word lock when it is set to 1 second (discussed in more detail in the next section).
  - Or `d.set_param(weight_file = 'weight file name (Bank added automatically)')` if you're trying to set the weight file for the beamformer.
- If you are running a scan without the scan coordinator:
  - `d.startin('sleep time', 'scan length')` where both parameters are set in seconds.
- If you are running a scan with the scan coordinator:
  - `from scanOverlord.scanOverlord import scanOverlord`
  - `sc = scanOverlord()`
  - `sc.addDealer(d)`
  - `sc.goLive()`
  - `sc.start_overlord()`

5 Putting things together

5.1 Initial tasks

There are currently two stable versions that we are using for production and testing of the backend. They are `sim_instance` and `instances4`. The version `sim_instance` is compatible with both the slow data rate packet_gen.m (used for simulated data testing) and full data rate roach simulator. There is still work remaining to be done to understand the BRAM structure of the data added to the roach for the roach simulator to be useful however, the framework is present. Therefore full data rate testing has been typically done on the production version of the code since they yield the same result. The simulator version is also compatible with the fits writer and thread_save methods in hashpipe.

All configurations of the backend system are handled through a configuration file interface found on the dibas directory path at 'etc/config'. For convenience there are 2 predefined versions of `dibas.conf` to help setup the system for the various versions. `dibas.conf.main` is the production version of the configuration and will provide all functionality for stable observations. Then there is `dibas.conf.pkt_gen`. This contains an extra flag in the default section of the config file for the backend to read in. This is the pkt_gen flag. With the flag enabled it will not arm the roaches and not interferer with pkt_gen.m With the flag disabled the roaches will be armed and whatever is in the roach simulator BRAMS will be sent. Interfacing either with the fits writer still requires updates to the conf file to either enable or disable dummy fits writer as well as applying the hashpipe save threads in BeamformerBackend.py.

To set a version of the backend system:

- Go to dibas directory: /home/groups/flag/dibas
- `./setVersion "version name"` (`sim_instance` or `instances4`)

Update to the corresponding dibas.conf:

- `cd etc/config`
- `cp dibas.conf."dersired_conf" dibas.conf`

After this, make sure to use a shell that has the change to the dibas or go to the python directory again as the setVersion command doesn't update the environment and symbolic link. If running the simulator, determine whether to dump files using the fits writer or save threads (Default behavior is to use the fits writer): In the newly applied conf file and using a text editor make changes under the section for the desired modes that will be tested.

At this point if running pkt_gen.m change the variables N_INPUT_BLOCKS to 4 in flag_databuf.h and WINDOW_SIZE to 2 in flag_net_thread.c

5.2 Running the system

Copy Mark Ruzindana's .bash_profile or just the aliases. It is located in /users/mruzinda/. This allows you to go to the correct directory with "gotopython" and start the banks with "startA", ... etc. Your .bash_profile is located in your home directory (/users/"your username"/).

Make sure that you set your files to be writable by the flag group with "`umask 0002`" in your .bash_profile as it is in Mark Ruzindana's. This is particularly important when trying to overwrite ".mat" files during post-processing among other things.

Also all users should get ssh keys set up so that they don't have to type in their password every time they ssh into a machine. It will also be a necessity when the dealer/player GUI is working. Do this by entering the following line into a terminal on any of the flag machines:

```
ssh-copy-id -i  /.ssh/id_rsa.pub username@flagmachine
```

For example, if I was on flag1 this would be:

```
ssh-copy-id -i  /.ssh/id_rsa.pub mruzinda@flag1
```

You don't have to do this without the dealer/player GUI (coming soon), but it helps not having to type your password every time you need to ssh into a different machine.

And when using MATLAB, as of right now (03/19/2019), GBO lost their MATLAB license so I have an alias in my .bash_profile for the BYU licensed MATLAB, `matlabpktgen`. Just type this into your terminal and it should open MATLAB. In the event that GBO's MATLAB license is renewed, the command is simply `matlab`. MATLAB should be opened on which ever machine you are operating with a VNC. In the case of the 2019 observations, it is flag1.

Dealer/player general notes: Bank A has the job of configuring the roaches. Therefore it uses the python module RoachDoctor to configure the roaches when setting a mode e.g `d.set_mode(FLAG_CALCORR_MODE)`. RoachDoctor reads in a text file on the dibas python library path (/home/groups/flag/dibas/lib/python) 'RC.txt'. This text file contains the flag representing the variable 'isConfigured'. As such, RC.txt with a 1 is interpreted as The roaches are configured. Dont program. And a file with 0 will result in the Roaches being programmed. If the roaches are reprogrammed, bit/byte/word lock is lost and has to be redone. So after programming the roaches, set RC.txt to 1. Follow the instructions below and you will know exactly when to this and everything else. Side note, When running packet_gen.m don't forget to program the roaches as well. Then change RC.txt to 1 after programming them.

DO NOT BIT/BYTE/WORD LOCK UNTIL THE INSTRUCTIONS TELL YOU TO DO SO. Explanation above.

AND READ THE ABOVE PARAGRAPHS BEFORE CONTINUING.

NOTE: Make sure that you record the scan numbers corresponding to a scan and corresponding to whether it is on source or off source during calibration. So you should have records of scan numbers that are on source and scan numbers that are off source. These

will be used in the `sensitivity_map.m` shown later in the section.

NOTE: Run a script in MATLAB by clicking on the green arrow around the top center of the MATLAB GUI.

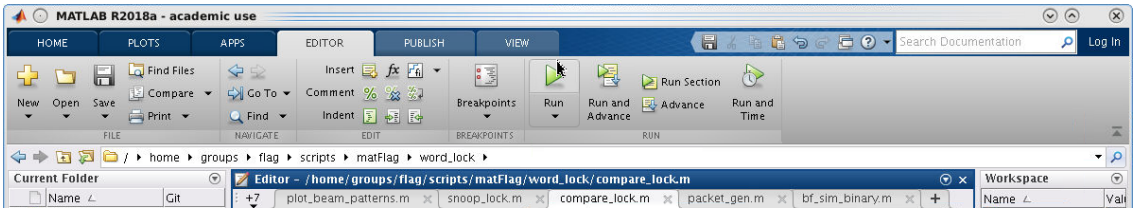


Figure 2: Button to push to run a MATLAB script.

Observation preparation: (1-7) Can be done prior to control of the LOs.

1. In the unlikely situation that there are many bad blocks on any player, reboot the flag machines.

2. Set RC.txt to 0. RC.txt is located in the dibas python library path

3. Start all 20 players, 4 per HPC. Go to dibas python library path (/home/groups/flag/dibas/lib/python) and:

   - If needed ssh into the machine (for example: `ssh flag2`)
   - When there, go to the dibas python library path (/home/groups/flag/dibas/lib/python)
   - Enter: `ipython player.py "bank name in capital letters e.g.  BANKA"` in the command line or use aliases from .bash_profile as suggested earlier in this section e.g. `startA`, `startB`, ...

   (/home/groups/flag/dibas/lib/python).

4. In a different terminal, go to the dibas python library path (/home/groups/flag/dibas/lib/python), start the dealer, and add all players to the dealer.

   - `import dealer`
   - `d = dealer.Dealer()`
   - `d.add_active_player(*d.list_available_players())`
     - Or `d.add_active_players('BANKA', 'BANKB')` if you're trying to start particular players.

5. Set the mode to CALCORR for calibration or any other relevant mode.

   - `d.set_mode('FLAG_CALCORR_MODE')`

- Or `d.set_mode('FLAG_RTBF_MODE')`, `d.set_mode('FLAG_PFBCORR_MODE')` if you're trying to set the real-time beamformer or fine channel correlator modes respectively.

6. Change RC.txt to 1. RC.txt is located in the dibas python library path (/home/groups/flag/dibas/lib/python).

7. (If HPCs are rebooted) Set the mode to CALCORR (or other mode) again since there are CUDA errors after rebooting HPC.

   At this point we are waiting for the LO to be put into Astrid/Scan Coordinator. As soon as it is, we can bit/byte/word lock.

8. Getting ready for bit/byte/word lock. Reboot the PAF.

   - Reset the netburner with the 'reboot_paf' batch file (must be done from windows computer with netburner software e.g. halibut). Access halibut using: `rdesktop halibut` from any FLAG machine. Change the username path from HALIBUT/'username' to ad/'username' before you type in your password. The batch file is located in the nb_kickoff folder provided by Luke Hawkins. Double click on the batch file and change the last 3 digits of the IP to 208 and click update.

9. Connect to the netburner, turn on the noise source and turn on the blades.

   - From ipython, enter the following under /home/groups/flag/dibas/lib/python/f_engine_config where the configuration scripts (f_engine_config_lo....py) are found or refer to the path when running the script:

     ```
     import corr, time, struct, sys, logging, socket, os
     from numpy import genfromtxt
     import numpy as np
     import nb_util as nb

     nb.connect()
     nb.set_rf_switch('NS')
     ```

   - From ipython enter the following (to turn on blades and set attenuation levels):

     ```
     nb.send_recv('PS0')
     nb.set_all_blades('9')
     ```

10. With noise source turned on, do bit lock.

- Open Labview GUI located in PAF/LabView/PAF - Monitor and Control provided by Luke Hawkins. Turn the GUI on using white arrow in top left corner for real-time analysis of time domain data and SBS data. Go to Time-domain tab for bit lock plots or SBS data for byte lock plots.
- From ipython enter the following (to set bit-lock - wait for outputs from the scripts to finish):

```
os.system("python f_engine_config.py flagr2-1  -n -f 0  -b 1 -i
0xaaaaaaaa")
os.system("python f_engine_config.py flagr2-2  -n -f 1  -b 1 -i
0xaaaaaaaa")
os.system("python f_engine_config.py flagr2-3  -n -f 2  -b 1 -i
0xaaaaaaaa")
os.system("python f_engine_config.py flagr2-4  -n -f 3  -b 1 -i
0xaaaaaaaa")
os.system("python f_engine_config.py flagr2-5  -n -f 4  -b 1 -i
0xaaaaaaaa")
```

11. Have Scan coordinator command the Test tone (LO 1B) to -10 bBm.
12. Turn test tone on and byte lock.
    - From Ipython enter the following (to set RF switch to TT (Test Tone)):

```
nb.set_rf_switch('TT')
```

    - Byte Lock: Set the Test Tone to be in one of the sidebands - the chosen default is between 10 & 50MHz above the LO (for example: LO=1450MHz, TT=1460MHz). The test tone can be set in the scan coordinator. The following script (called the accelerator script) can be found in /home/groups/flag/dibas/lib/python/f_engine_config and can be run in ipython with these commands:
      - Run scripts:

```
os.system("twc_byte_lock.py A4 A6 --commit")
os.system("twc_byte_lock.py B5 B6 B7 --commit")
os.system("twc_byte_lock.py C5 --commit")
os.system("twc_byte_lock.py D1 D3 D4 D6 D8 --commit")
os.system("twc_byte_lock.py E1 E4 E6 --commit")
```

      - The channels specified above (e.g A4, A6, B5, B6, etc) are ones that were not byte locked and need to be byte locked.

- So if the plots in the SBS data tab of the LabView GUI have spikes on both sides of the spectrum, add the channel to the argument of the command. The command is run for each blade as seen above. So one command per blade of which there are 5 (A, B, C, D, and E).
- Bytes should be aligned now with one spike on the appropriate side of the spectrum. For eaxmple, if the LO=1450 MHz and the TT=1460 MHz, there should be one tone 10 MHz to the right of the center.
- If the channels selected aren't byte locked, you can manually byte lock by selecting the appropriate channel and clicking swap bytes on the left of the LabView GUI under the SBS data tab. Or contact Luke Hawkins for help.

13. Have Scan coordinator command the test tone to -60 dBm. Effectively turning it off.

14. If word lock is going to be done, follow 15 - 20 ('WORD LOCK' placed at the beginning of the instruction for easy identification), otherwise, turn off the noise source and go to instruction 21 onward:

```
nb.set_rf_switch('OFF')
```

15. WORD LOCK - (Word lock is optional depending on time constraints and whether calibrated weights will be used for multiple observations) If word lock is being done, turn noise source on:

```
nb.set_rf_switch('NS')
```

16. WORD LOCK - Set int_length to 1s perform 1s scan for word lock: `d.set_param(int_length=1)` then `d.startin(5,1)` and record the time stamp of the FITS file.

17. WORD LOCK - Open MATLAB and use snoop_lock.m located in /home/groups/flag/scripts/matFlag/word_lock to word lock and record the reference element shown in the command window after the script is run.

    - Change the time stamp variable (tstamp) to the time stamp from the FITS file in the 1s scan.
    - Record the reference element shown in the command window.
    - Make sure the plots have data. If they don't, then either you typed the time stamp in snoop_lock.m wrong or the fits files weren't generated.
    - Make sure the test tone is not in the plots and that the phase (bottom plot of sub plot) is somewhat linear across most of the elements. This maybe harder to tell if you are inexperienced so skip this step and continue or ask a more experienced person.

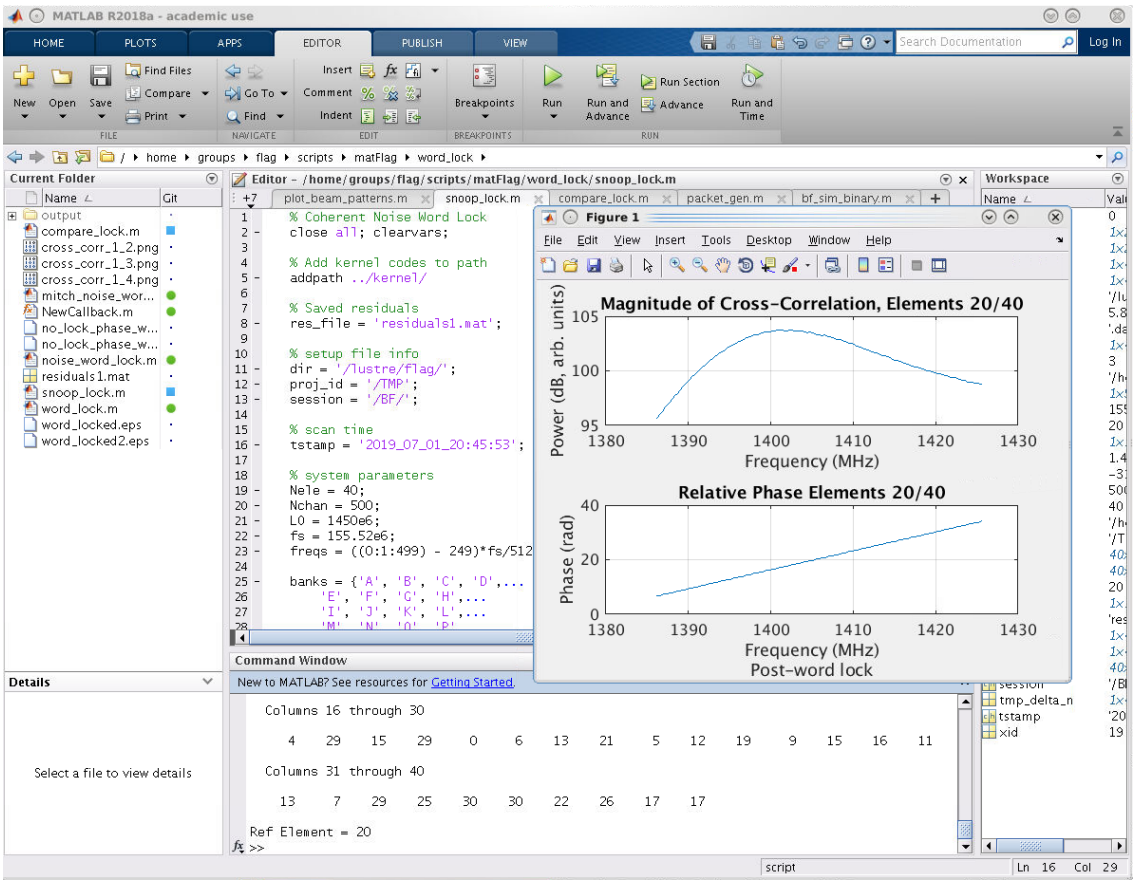


Figure 3: An example of the plot and output of the command window from the snoop_lock.m file. And the time stamp needs to be changed on line 16 of snoop_lock.m.

18. WORD LOCK - Apply timestamp to set_delays.py located in the dibas python library path (/home/groups/flag/dibas/lib/python) by changing/modifying the `delay_src` variable and run.

19. WORD LOCK - Run another 1s scan (`d.startin(5,1)`), record the time stamp of the FITS file, and in MATLAB, verify word lock with compare_lock.m using the reference element from snoop_lock.m.

    - Change the time stamp variable (tstamp) to the time stamp from the fits file in the second 1s scan after changing and running set_delays.py.
    - Change the reference element to the recorded one by changing the `ref_el` variable at the bottom of the file.
    - If the result in the command window is all zeros, then you have perfect word lock. If there are a few (2 or so) offsets by 1 or -1 it should be okay. But if there

are a variety of different integers, then you have to reprogram the roaches and do bit/byte/word lock again.

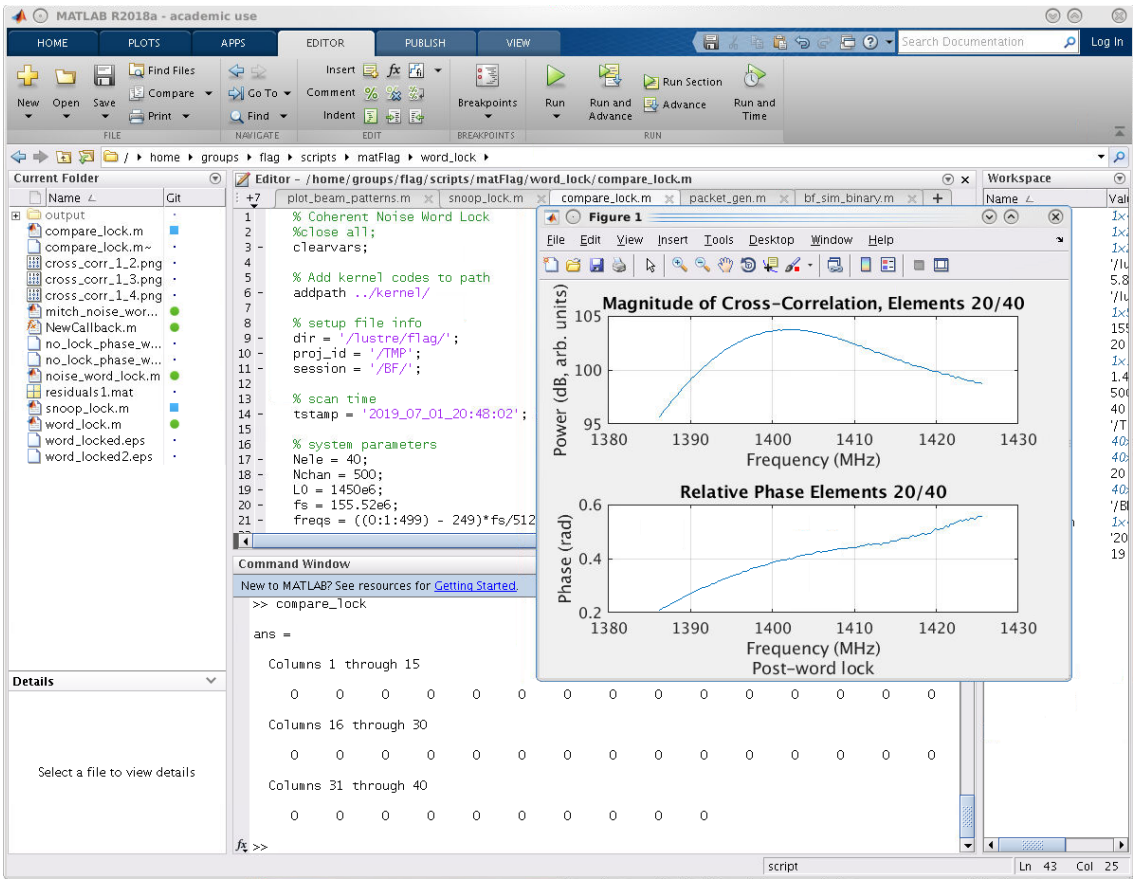


Figure 4: An example of the plot and output of the command window from the compare_lock.m file. And the time stamp needs to be changed on line 14 of compare_lock.m.

```
46      % compute delays relative to first element to determine max delay
47 -    ref_el = 20;
48 -    tmp_delta_n = compute_delay(R, freqs, fs, ref_el);
49 -    round(tmp_delta_n)
```

Figure 5: An example of the line of code you need to change for the reference element which is line 47 in compare_lock.m.

20. WORD LOCK - If word lock is done, close MATLAB and turn the noise source off.

```
nb.set_rf_switch('OFF')
```

21. Set the quantization gain to ensure that it is set correctly (20 for CALCORR and RTBF & 10 for PFBCORR). In the python library path (/home/groups/flag/dibas/lib/python), do this by:

    - Checking the set_quantgain.py file to see/change the setting of the `quant_gain` variable.
    - Set it appropriately then run the set_quantgain.py file by: `python set_quantgain.py`.

22. Set parameters for the dealer as to prepare it to be handed over to the scan overlord. Go to next subsection for specific instructions on different modes (5.2.1 - CHECKLIST FOR EACH MODE). 22 - 24 provide you with general instructions for all modes if you are familiar enough with the system so go to section 5.2.1 if you want more precise ones.

    - `d.set_param(int_length = 0.5)` which is typically 0.5 unless performing word lock when it is set to 1 second (discussed in more detail in the next section).
        - Or `d.set_param(weight_file='weight_file name with "_.bin"')` if you're trying to set the weight file for the beamformer. Bank is added to the file name automatically.
        - Or `d.set_param(channel_select=1)` when running PFBCORR mode to select bins containing HI.

23. Import overlord module: `from scanOverlord.scanOverlord import scanOverlord`.

24. Create overlord and connect:

    - `sc = scanOverlord()`
    - `sc.addDealer(d)`
    - `sc.goLive()`
    - `sc.start_overlord()`

At this point we are ready for commands to come through Scan Coordinator and observe! After observations, turn off the blades and disconnect the netburner:

```
nb.send_recv('PS1')
nb.NB.close()
```

5.2.1 CHECKLIST FOR EACH MODE

CALCORR mode:
Quick reference checklist for CALCORR mode starting with setting the mode so you already set the dealer and added the players. The first set of commands are used when the receiver is on telescope (on the GBT) and the second set of commands are used when the

receiver is in the OTF (in the OTF).

On the GBT:

- `d.set_mode('FLAG_CALCORR_MODE')`
- `d.set_param(int_length=0.5)`
- `from scanOverlord.scanOverlord import scanOverlord`
- `sc = scanOverlord()`
- `sc.addDealer(d)`
- `sc.goLive()`
- `sc.start_overlord()`

In the OTF:

- `d.set_mode('FLAG_CALCORR_MODE')`
- `d.set_param(int_length=0.5)`
- `d.startin(5,10)` Sleep time and scan time can be set to whatever the user wants. (5,10) are just examples.

PFBCORR mode:
Set the integration length to 0.5s and the quantization gain to 10. For HI observations the channel select needs to be set to 1.

Quick reference checklist for PFBCORR mode starting with setting the mode so you already set the dealer and added the players. The first set of commands are used when the receiver is on telescope (on the GBT) and the second set of commands are used when the receiver is in the OTF (in the OTF).

On the GBT:

- `d.set_mode('FLAG_PFBCORR_MODE')`
- `d.set_param(channel_select=1)`
- `d.set_param(int_length=0.5)`
- Run set_quantgain.py in another terminal under '/home/groups/flag/dibas/lib/python' by typing '`python set_quantgain.py`'. Make sure the quant gain is set to 10 in the file.
- `from scanOverlord.scanOverlord import scanOverlord`

- `sc = scanOverlord()`
- `sc.addDealer(d)`
- `sc.goLive()`
- `sc.start_overlord()`

In the OTF:

- `d.set_mode('FLAG_PFBCORR_MODE')`
- `d.set_param(channel_select=1)`
- `d.set_param(int_length=0.5)`
- Run set_quantgain.py in another terminal under '/home/groups/flag/dibas/lib/python' by typing '`python set_quantgain.py`'. Make sure the quant gain is set to 10 in the file.
- `d.startin(5,10)` Sleep time and scan time can be set to whatever the user wants. (5,10) are just examples.

RTBF mode (Pulsar/transient scan):
Run CALCORR mode first with telescope on bright calibrator to generate beams. Then generate the weights with the instructions in the post processing section below.

In short, open MATLAB and open files `sensitivity_map.m`, `scan_table.m`, and `plot_beam_patterns.m` found in \home\groups\flag\scripts\matFlag under \sensitivity_maps\, \kernel\, and \patterns\ respectively.

Make the changes shown in the post-processing section below in each file and run `sensitivity_map.m` to generate '.mat' weight files, then run `plot_beam_patterns.m` to plot the beam patterns and generate '.bin' files used by RTBF mode. These files are saved in \home\groups\flag\weight_files\.

BEFORE WEIGHT GENERATION (POST-PROCESSING STAGE), if MATLAB scripts don't create folders for you:

- Create a directory called 'mat' under /lustre/flag/"session name"/BF/ e.g. /lustre/flag/AGBT19A_407_05/BF/mat.
- And also create a directory with the session name for saving beam pattern figures with `plot_beam_patterns.m` under /home/groups/flag/scripts/matFlag/patterns/pattern_plots/ e.g. /home/groups/flag/scripts/matFlag/patterns/pattern_plots/AGBT19A_407_05/.

Quick reference checklist for RTBF mode starting with setting the mode so you already set the dealer and added the players. The first set of commands are used when the receiver is on telescope (on the GBT) and the second set of commands are used when the receiver is in the OTF (in the OTF).

On the GBT:

- `d.set_mode('FLAG_RTBF_MODE')`
- `d.set_param(weight_file='weight_file name with "_.bin"')`
- `from scanOverlord.scanOverlord import scanOverlord`
- `sc = scanOverlord()`
- `sc.addDealer(d)`
- `sc.goLive()`
- `sc.start_overlord()`

In the OTF:

- `d.set_mode('FLAG_RTBF_MODE')`
- `d.set_param(weight_file='weight_file name with "_.bin"')`
- `d.startin(5,10)` Sleep time and scan time can be set to whatever the user wants. (5,10) are just examples.

TURNING OFF BLADES AND DISCONNECTING NETBURNER:
After observations, turn off the blades and disconnect the netburner:

```
nb.send_recv('PS1')
nb.NB.close()
```

5.2.2 DEBUGGING IN CASE OF STALLS OR TOO MANY BAD BLOCKS

- If you see stalls in the terminals with the print statement, "NET: HANGING HERE!!!" abort the scan. Then you will need to kill that terminal (Ctrl+c and/or Ctrl+Z). If you still see the statement scrolling down then close that terminal and reopen it.
- Make sure you ssh to the right machine. The desktops are in order of flag machine in case you didn't see that.
- Then type '`ps aux | grep hashpipe`' to see whether that hashpipe process is still running. And do the same with ipython, '`ps aux | grep ipython`'.

- Then kill the process by typing '`kill -9 "hashpipe process id"` ' and/or '`kill -9 "ipython process id"` '.
- Then restart the player and reset the mode and run a couple of 10 minute dummy scans with "`d.startin(5,10)`".
- After this, you can do all of the scan overlord stuff.
- If there are too many banks stalling (without the above net hanging here statement) or too many terminals with bad blocks then abort the scan and restart those players.
- Notes on operation:

  While the scan overlord is running, in the event of many hanging banks or another reason to abort a scan wait for the abort to propagate through the system. Then, a SIGINT (Ctrl+c) can be issued at anytime to recover the dealer and again be able to issue commands to the players. This will typically fix any issues in the system. If the hangs or problems are persistent issue another abort through the Scan Coordinator, wait for the banks to end, interrupt the overlord and issue a SIGINT to all 20 banks and restart them. Run a dummy scan (`d.startin(5,10)`) for the desired mode and then re-initialize the scan overlord, connect the dealer and start listening. This had fixed all issues in the first session from the July/Aug commissioning to get on track quickly.

  In the event of a loss of bit/byte lock that leads to a new word lock solution, it's important to remember that the delays from the previous word lock solution are still active. Therefore, the delays have to be reset to 0. Reprogramming the roach is sufficient or the set_delays python file can be extended to reset the delays.

Helpful files:
set_delays.py: run to update word_lock file. Edit timestamp on the third line.
set_quantgain.py: run to update the desired quant gain. Change the desired value on the third line. The default value when the roaches are configured is 20 This will be helpful when changing between CALCORR where we are to operate at 20 and PFBCORR where we are to operate at 10.

6 Post-processing

The post-processing code is all done in MATLAB and it is currently being written in python as well. This section will be detailed in important files and provide a brief description of some of the other files. There are 3 important files used during observations. These files are; `sensitivity_map.m`, `scan_table.m`, and `plot_beam_patterns.m` found in \home\groups\flag\scripts\matFlag under \sensitivity_maps\, \kernel\, and \patterns\ respectively.

\sensitivity_maps\\`sensitivity_map.m` uses the covariance matrices obtained from a scan using the coarse correlator operational mode, and calculates and plots the sensitivity over the field of view and also plots the system temperature over antenna efficiency. It is typically used after a calibration scan to analyze the data and generate weights. It stores these weights as well as steering vectors as ".mat" files. To use this `sensitivity_map.m` file, the user will need to add a few things. The user will need to add the session, the scan numbers, the source, a variable that integrates over the samples in the data, and the type of observation/grid e.g. full grid or seven-point grid. The figure below shows an example of the lines of code that would need to be added.

```
% AGBT16B_400_12 Seven Beams %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% session = AGBT16B_400_12;
% Calibrator
% on_scans = [131:137];
% off_scans = [130, 138];
% off_scans = [130];
% source = source3C48;
% Nint = -1;
% note = 'seven';

% AGBT16B_400_12 Grid %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
session = AGBT16B_400_12;
% % Calibrator
on_scans = [32:36, 38:42, 44:48, 50:54, 56:60, 66:70, 72:85];
off_scans = [37, 43, 49, 55, 61, 71];
source = source3C295;
Nint = 2;
note = 'grid';

% AGBT16B_400_13 Grid %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% session = AGBT16B_400_13;
% % % Calibrator
% on_scans = [19:23, 25:29, 31:35, 38:42, 44:48, 50:54, 56:59];
% off_scans = [24, 30, 36, 43, 49, 55];
% source = source3C123;
% Nint = 2;
% note = 'grid';
```

Figure 6: An example of the lines of code that would need to be added.

In Figure 2, you can see that recording the scans is important (On source vs. Off source and scan number). The user must look at the file \kernel\\`source_table.m` to make sure that the source being scanned is in the table. This source is used to calculate the source flux density which is used to calculate the sensitivity. There are 5 parameters associated with each source and these parameters are/were acquired from the astronomers in the project. The `Nint` variable is the number of samples that are integrated over. If it is set to -1, then all of the samples are integrated. The `note` variable labels the relevant files so that the user knows what kind of calibration the file belongs.

\kernel\\`scan_table.m` has all of the parameters associated with a particular session.

The figure below shows the lines of code that would need to be added to `scan_table.m` for each session. So just copy the previous sessions lines and modify what needs to be changed.

```
%% AGBT18A_443_01
% % Scan numbers and time stamps
AGBT18A_443_01.session_name = 'AGBT18A_443_01';
% log_filename = sprintf('%s/%s_01/ScanLog.fits', meta_root, proj_id_5);
log_filename = sprintf('%s/%s_01/ScanLog.fits', meta_root_array{1}, proj_id{5});
scan_data = fitsread(log_filename, 'bintable', 1);
time_stamps = unique(scan_data{1});

for i = 1:length(time_stamps)
    my_stamp = time_stamps{i};
    my_stamp = strrep(my_stamp, '-', '_');
    my_stamp = strrep(my_stamp, 'T', '_');
    AGBT18A_443_01.scans{i} = my_stamp;
end

% Frequency mask
bad_freqs = [];
AGBT18A_443_01.bad_freqs = bad_freqs;

% Element mappings
X = [1:7, 35, 9:12, 14, 13, 15:19];
Y = [21, 20, 23:34, 36:38]; % [21, 20, 23:34, 8, 36:39];
goodX = X; % goodX(X < 9) = [];
goodY = Y;
% goodY(Y==33) = [];

AGBT18A_443_01.X = X;
AGBT18A_443_01.Y = Y;
AGBT18A_443_01.goodX = goodX;
AGBT18A_443_01.goodY = goodY;

AGBT18A_443_01.fudge = 0;
AGBT18A_443_01.Xdims = [-0.23, 0.23];
AGBT18A_443_01.Ydims = [-0.23, 0.23];

AGBT18A_443_01.beam_el = [ 3.94, 3.94,  0.0, 0.0, 0.0, -3.94, -3.94]*1/60;
AGBT18A_443_01.beam_az = [-2.275, 2.275, -4.55, 0.0, 4.55, -2.275,  2.275]*1/60;
```

Figure 7: An example of the lines of code that would need to be added to `scan_table.m` for each session.

`meta_root_array{1}` and `proj_id{5}` are the directory \home\gbtdata\, and the project ID (for example AGBT18A_443) respectively. The project id is set close to the top of the file. Just add the ID to the array in the figure below. Then take note of the index in the array and use it in the `log_filename` string.

```
25 -    proj_id = {'AGBT16B_400', 'AGBT17B_360', 'AGBT17B_221', ...
26          'AGBT17B_455', 'AGBT18A_443', 'AGBT19A_407', ...
27          'AGBT19A_221', 'AGBT18B_358', 'AGBT19A_091', ...
28          'AGBT19A_365', 'AGBT19A_116'};
```

Figure 8: The array of the project IDs in the `scan_table` file. Just add the new project ID to this array.

The `X` and `Y` variables are the X and Y polarization elements. `Xdims` and `Ydims` are the minimum and maximum values on the sensitivity map plot. The last important variables to take note of are the `beam_el` and `beam_az` variable. They are the elevation and cross-elevation coordinates of the seven beams that are used in the weights files for the real-time beamformer. These are measured in arc-minutes and the offsets must be acquired from the astronomer/user that wrote the astrid scripts used to control the telescopes movement during a calibration scan.

The final important file used during a scan is the \patterns\`plot_beam_patterns.m`. This file plots the beam patterns and more importantly generates the binary weight file used in the real-time beamformer. The figure below shows the lines of code that would need to be modified in this file. The name of the weight file should NOT exceed 8 characters so modify and uncomment line 19 in the figure (so comment line 17 & 18).

```
12      % Desired session
13 -    session = AGBT18A_443_01; %AGBT17B_455_01; %AGBT16B_400_12;
14 -    note = '1st_seven'; % 'grid';
15
16      % Output filename
17      % rtbf_filename = sprintf('%s/%s/BF/mat/w_%s_%s.bin', data_root,...
18      %       session.session_name, session.session_name, note);
19 -    rtbf_filename = sprintf('%s/w443_01.bin', weight_dir);
```

Figure 9: An example of the lines of code that would need to be modified in the `plot_beam_patterns.m` file.

This file uses a function called `create_weight_file` to generate the binary weight files. These weight files are generated for each bank (A to T). One variable that might need to be modified in the `sensitivity_map.m` and `plot_beam_patterns.m` as well as other files is the LO frequency variable, `LO_freq`. This variable is in megahertz so typically it is set to 1450, but the observer and FLAG backend user will need to coordinate to set this parameter.

There are plenty of other files that are used during scans, but these are the files used during observations. Other files can be investigated by the user. A brief summary of some of the rest of the post-processing code are listed below:

- `extract_covariances.m` - This is a function that extracts covariances from the FITS files generated by HASHPIPE when using the correlator operational mode.

- `get_antenna_positions.m` - A function used to calculate the elevation and cross-elevation offsets. These help determine the position of the beams on the grid.
- `get_steering_vectors.m` - A function used to generate steering vectors.
- `get_grid_weights.m` - A function used in plot_beam_patterns.m that selects 7 beams from the weights generated by sensitivity_map.m. The total number of beams correspond to the total number of on-pointings on the grid.
- `get_element_patterns.m` - A file that plots the element patterns using the steering vectors generated by sensitivity_map.m
- `RTBF_data_analysis.m` - A file that plots output of the real-time beamformer. Provides intensity plots for short time integration (STI) windows vs. frequency bins. It also plots the power over the STI windows.
- `bf_tf_power.m` - Also plots output of the real-time beamformer operational mode. Plots power over the frequency bins
- `extract_bf_output.m` - Extracts the data from the FITS files generated by the real-time beamformer. These files were too large to be read by the fitsread() function in MATLAB. Low-level functions were used to interact directly with CFITSIO library functions. These functions allow the user to read a fraction of the data from the fits files rather than the entire file.
- `covariance_test.m` - This file plots the spectra generated with the covariance matrices from the coarse correlator operational mode. This is done to analyze the data over the bandwidth.

References

[1] "Dibas 1.0 documentation." http://www.gb.nrao.edu/~mwhitehe/dibas/html/index.html. Accessed: 2017-01-23.