

WQ7T Enhanced Version of FreqShow – Panadapter Installation Setup and Operations Manual

!VERY IMPORTANT!

Please note, the enhancements to FreqShow by WQ7T require the Python-Scipy Library in addition to the original dependencies for FreqShow. The Python-Scipy Library will have to be installed for these new versions of FreqShow to work, otherwise the program will not load.

Original dependencies required by Adafruit/FreqShow

<https://learn.adafruit.com/freq-show-raspberry-pi-rtl-sdr-scanner/installation>

install scripts:

```
sudo apt-get update
```

```
sudo apt-get install cmake build-essential python-pip libusb-1.0-0-dev python-numpy git  
pandoc
```

<https://www.scipy.org/install.html>

install script :

```
python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose
```

OR

alternate install script:

```
sudo apt-get install python-numpy python-scipy python-matplotlib ipython ipython-notebook  
python-pandas python-sympy python-nose
```

INDEX

page 1

SECTION 1

Main Screen Operation	page 3
Main Settings Screen operation	page 4
Making Your Own Default Settings to FreqShow Permanent	page 9

Section 1 – Operating the Panadapter

Overview

1. The original version of **FreqShow**, although I thought it was very nice and well written, the code to me seemed very clean! However it did not have enough functionality for me. So that's what motivated me to add it. It's also the program I stumbled across first. Even though I didn't know the Python programming language, and I still have a lot to learn, I managed mainly by trial and error to get most of this working, fairly bug free. I still think there are things I can improve on, but it works pretty well the way it is at this point and the enhanced version has plenty of controls to display the 1st IF of your radio without being too difficult. Almost all of the operating parameters can be changed while the program is running, and it has the ability to zoom well into nice detail. You can get 10kHz full resolution on the 2.8" version, 320 pixel width and 25kHz full resolution using the 7" version 800 pixel width without stuffing pixels. You can also, of course, scan wider frequency spans for use as an over-all band scope, but I think the ability to zoom at full resolution with this enhanced version of FreqShow is where this program really shines. You can also choose to do either FFT averaging, or FFT peaks both with an adjustable array depth. I also added the ability to choose between 9 different pre FFT windowing functions including the Kaiser function with an adjustable Beta so you really get to tweak the result to your own liking. You can also choose between graph or waterfall outputs to the screen.

!VERY IMPORTANT!

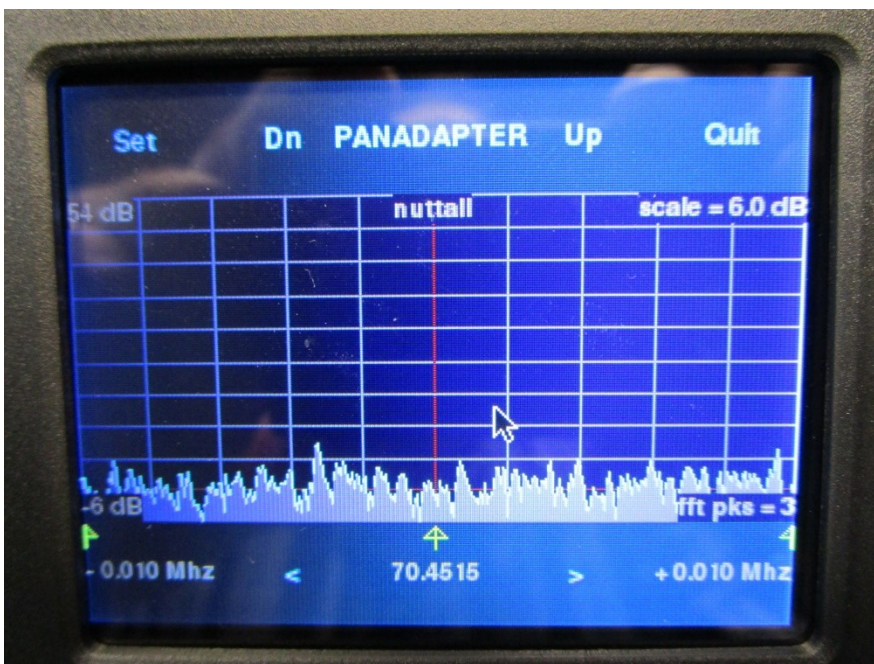
Please note, the enhancements to FreqShow by WQ7T require the **Python-Scipy Library. The Python-Scipy Library will have to be installed for these versions of FreqShow to work, otherwise the program will not load.**

There are also desktop launcher files in the FreqShow_Large and FreqShow_Small directories that can be copied directly to the Desktop.

My 2.8" version is shown, but the 7" version is the same except for differences noted!

FreqShow – WQ7T enhanced version

1. Main screen operation



Set - This button accesses the main settings screen where you choose the parameter to change.

Dn – This button scrolls both upper and lower dB limits on the grid down by 5dB every time you select it. *This control is an enhanced version addition.*

PANADAPTER – This is a button and part of the original FreqShow, it toggles between the graphical view and the Waterfall view.

Up - This button scrolls both upper and lower dB limits on the grid up by 5dB every time you select it. *This control is an enhanced version addition.*

Quit – Selecting this button takes you to next screen to confirm that you want to exit the program

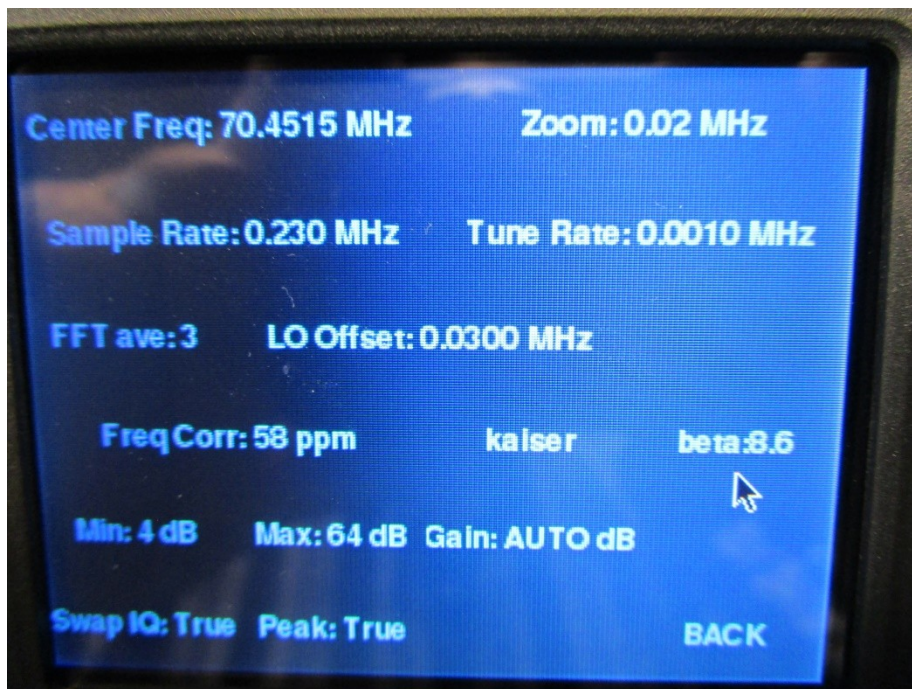
< - This button decreases the Center Frequency by the Tune Rate setting – default is 1kHz. *This control is an enhanced version addition.*

> - This button increases the Center Frequency by the Tune Rate setting – default is 1kHz. *This control is an enhanced version addition.*

Graphical Area or Grid – This actually functions as a button to toggle this area between this view or to fill the full screen. This option was part the original FreqShow as well.

Note: The other text on the screen is only informational and not selectable.

2. Main Settings Screen operation



Each one of the above parameters are selectable to allow changing the value to tweak the Pan-adapter to your liking within limits of course. When you initially access this screen via the Set button on the Main Screen this will show you the current parameters being used.

Note: the “beta” parameter only shows up on the screen when the Kaiser pre-FFT windowing function is selected.

To select any of these parameters, simply use either the mouse or the touchscreen.

BACK – returns you to the Main Screen with any new values you changed, which will now be shown on this screen.

Center Freq: – this is going to be the Frequency in the very middle of the display – in the case of Pan-adapter use, should be your 1st IF frequency, in MHz

Zoom: - This is the span in MHz across the screen – there are limits to this, and the limits depend on the “Sample Rate:” setting and the width of the screen in pixels. This is also dependent on the Sample Size which is fixed at 8192 for the program. 8192, (*which has to be a power 2 for the FFT to execute efficiently*), was chosen specifically to allow reasonable speed using the Raspberry Pi and at the same time give decent zooming ability. When I tried using a Sample Rate of 16384 the Raspberry Pi can do it, and you can zoom in twice as narrow, but it slows down noticeably, so we left it at 8192.

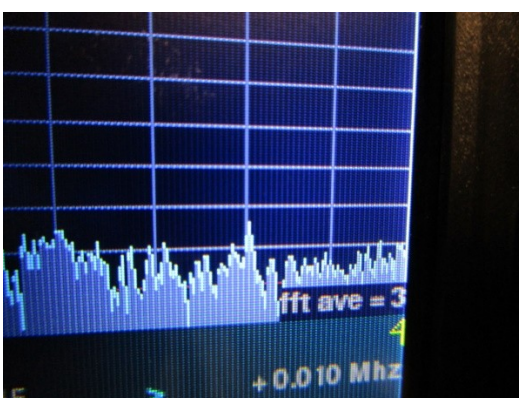
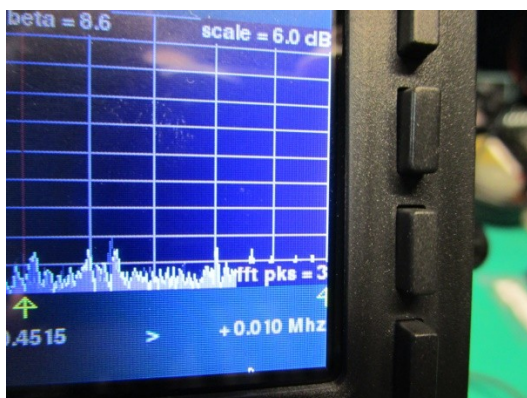
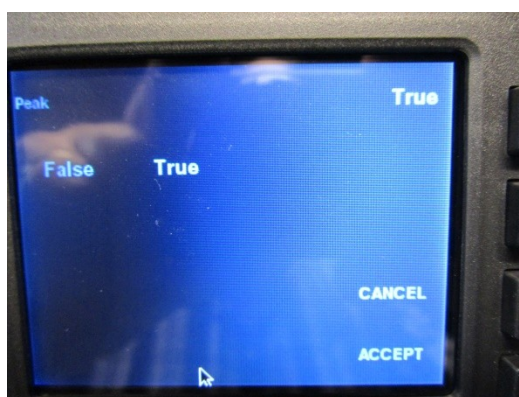
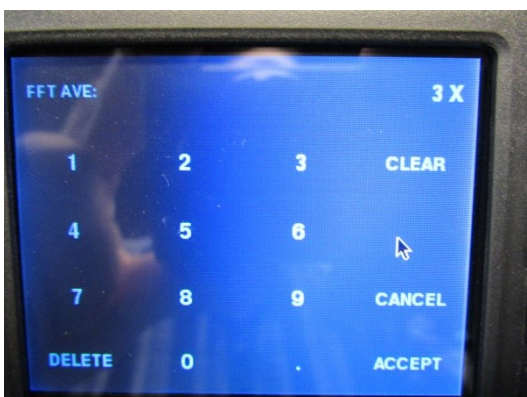
For example : When the Sample Rate: is set to 0.230 MHz, the lower Zoom limit, will be 0.01MHz (10kHz) for the 2.8” screen with a width of 320 pixels and 0.025MHz (25kHz) for the 7” screen with a width of 800 pixels. If you try to set either of these screens to a narrower span than the limit, the program will automatically reset the Zoom value to equal the Sample

Rate, so it doesn't crash the program. The upper zoom limit of course is equal to whatever the current Sample Rate is. There are ways of exceeding that upper limit, but I did not incorporate that ability in the program code. *This control is an enhanced version addition.*

Sample Rate: - This value is limited by the RTL2832 820T/820T2 device. There are two ranges of acceptable values, **225001 to 300000 Hz** and **900001 to 3200000 Hz**. Because we are truncating the value to 3 places after the decimal, the ranges will be, in the program, **0.226 to 0.300 MHz** and **0.901 to 3.200 MHz**. *The reason I chose 0.230 as the default, it allows the lower limit Zoom settings of either 10kHz or 25kHz depending on the display you are using. If you try to set the value to something outside of these ranges, the program won't change it.*

Tune Rate: - This is the step size used by the < and > buttons on the Main screen. For some reason the RTL device I did all of my development with, always truncated the Center Frequency to the nearest kHz, so trying anything less than 0.0010 Mhz wouldn't change the Center Frequency. 0.0010 Mhz works though! I still need to investigate that further! *This control is an enhanced version addition.*

FFT ave: - This must be an integer value and at least 1, if you try to set this to 0, it will reset it to 1. This sets the array depth for calculating either the average value for each frequency bin or grabbing the peak value for that frequency bin over this number of successive scans. This works in conjunction with the Peak: True or False setting. **Note: Peak = False gives you averaging, Peak = True gives you peak persistence.** *Note: do not use the decimal point on the setting screen, if you do, the program will terminate and you will have to restart it via the desktop launcher.* *This control is an enhanced version addition.*



LO Offset: - Although the original FreqShow incorporated the code to eliminate the DC spike at the Center Frequency setting of the RTL device, there is still some LO feed through, from the RTL device, that can be seen as a “pip” that doesn’t move as you tune your rig. It will stay in the center of the graph. So we needed a means to shift that to one side or the other out of the display so you wouldn’t see it. It’s kind of irritating to always see a small pip right there in the middle of the span, so this is how we dealt with it. It really isn’t gone, it’s just shifted far enough so that it’s beyond what we are displaying on the screen. The ability to do this somewhat depends on other settings on the pan-adapter of course. When we have a valid zoom set, the program trims off the frequency bins* that we do not need, and then only displays what we want on the screen. There has to be enough room within the trimmed frequency bins to contain the center frequency pip in order for this to work. My program code checks to see if there is room enough to shift the actual RTL center frequency by the value of the setting and if there is, it will do it. If you try to set the LO Offset too far away, the code will see that and not apply the shift. If you set the LO Offset not far enough, of course you will still see it, but shifted in the direction you told to go from the center. You can recognize it as a carrier that doesn’t move, when everything else does. *This control is an enhanced version addition.*

For example: If the Zoom is set to equal the Sample Rate we will not be able to trim any frequency bins*, so the LO Offset will be ignored and most likely you will see a small pip in the middle of the graph!

*Each pixel represents one frequency bin

FreqCor: - in ppm - This sets the correction in ppm of the xtal osc in the RTL device. If we were demodulating a carrier this becomes really important even with the newer enhanced Nooelec RTL .5ppm TCXO devices. ***However a setting of 0 ppm here will suffice for most of the newer enhanced RTL .5ppm TCXO devices for our purposes. Note: do not use the decimal point on the setting screen, if you do, the program will terminate and you will have to restart it via the desktop launcher.*** *This control is an enhanced version addition.*

If you are using an unenhanced non TCXO RTL device it will need this correction. The original RTL2832’s that I purchased were direct from Hong Kong, China and unenhanced. As you can see I was able to determine that this one needed a correction of 58ppm.

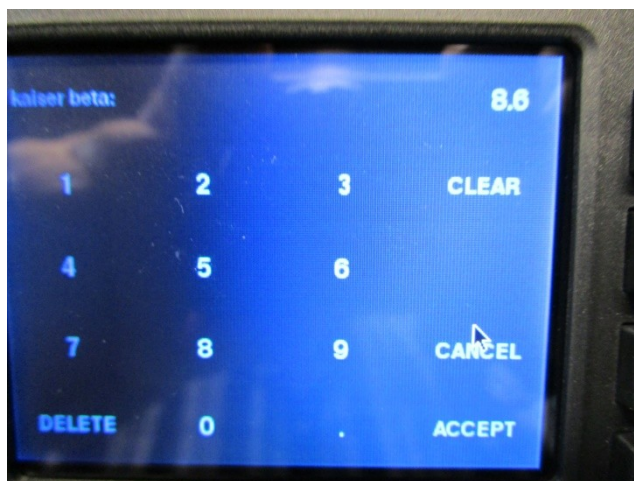
To see what the correction should be in ppm, you will need to disconnect the RTL-SDR from your 1st IF and connect it to an antenna. You will need to have a Linux Terminal window opened, preferably via the Putty program, and type “sudo rtl_test -p” without quotes at the command prompt. Observe the output text, if your RTL dongle has been on long enough to stabilize in temperature the ppm value should be fairly consistent. Write this value down and this is what you’ll set this parameter to.

Kaiser – This is one of the 9 pre-FFT windowing choices available. Clicking on this setting will allow you to choose the windowing type used on each individual frequency bin before the FFT is applied. The FFT then converts the 8192 time domain samples per bin to a frequency

domain value. The windowing function effectively removes, not perfectly though, power from the side-lobes on either side of the frequency that the bin represents, so that the subsequent application of the FFT will more accurately represent the actual power for that frequency bin.

Hope that makes sense! [Windowing Functions Improve FFT Results, Part I | EDN](#)

Because there are 9 different windowing functions, you can try the different ones to see if there is one you prefer. The Kaiser function has an additional parameter called the “**beta**” a value anywhere between 0 and 100 that allows it to emulate some of the other windowing functions. It is there in case you want to experiment with it. The default is “**nutall**”. *These controls are an enhanced version addition.*

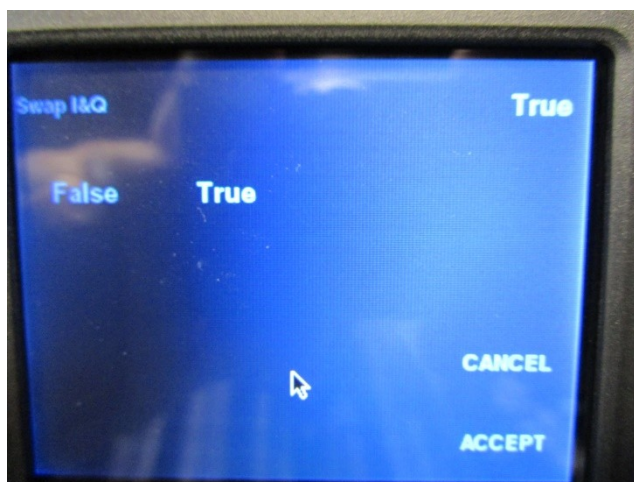


Min: Max: and Gain: - These three individual buttons set the level values for the Main Screen grid. Although you can set both min and max to “AUTO” I don’t recommend it! I like to leave a 60 dB difference between the min and max values as that’s about all the difference the RTL dynamic range is good for. I generally leave the Gain in “AUTO” though.



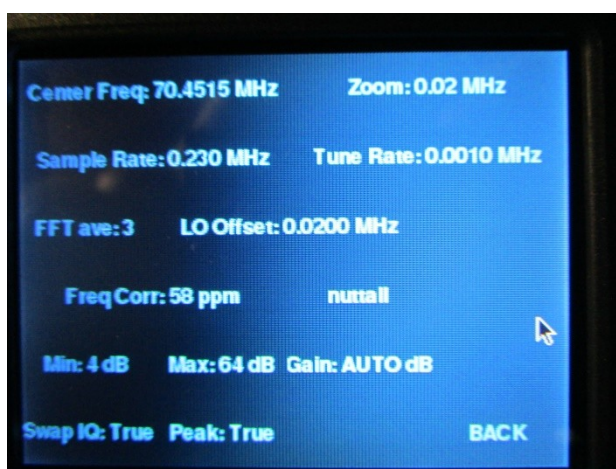
Setting Min to “AUTO” can result in some strange behavior, I will eventually explore that, but just avoiding that setting for Min solves the issue. *These controls are virtually unchanged from the original FreqShow.*

Swap I Q: - This setting allows the display of the frequency span to be reversed. Instead of the “lower frequency on the left side to higher frequency on the right side” which would be the normal way to display the scan, setting **Swap I Q: = True** reverses that, putting “*higher frequency on the left side to lower frequency on the right side*”. This ability will become necessary if your 1st IF frequency is the lower side image from your 1st LO in your transceiver. That is the case with my rigs and I think typical, but not always the true. *If you have this set wrong, but your Center Frequency is set correctly to your 1st IF, as you tune your radio's Frequency dial, the carriers will move in the direction opposite to what you want and it tends to be somewhat confusing.* Set correctly things will seem much more natural. *This control is an enhanced version addition.*



ACCEPT - click this button on all sub setting screens to “accept” the change, it will take you back to the Main Settings Screen where you can continue to make other changes.

When you are finished making changes and back on the Main Settings Screen, “Back” returns you to the Main Screen with the new settings.



IMPORTANT: These settings will remain as long as the program remains running, however the next time the FreqShow program starts, it will again load the default values not your changes.

So the next section will show you how to set your own program defaults!

Step 5: The next thing we want to do is backup the folder that has the file we need to modify, so depending on the version you have the name of the folder will either be **FreqShow_Small** or **FreqShow_Large** and you should see it in blue in the returned list.

Here is the command to type at the prompt and it depends on which version you have

Where you see the it's a single space

```
sudo cp -r FreqShow_Small FreqShow_Small_bkp
```

OR

```
sudo cp -r FreqShow_Large FreqShow_Large_bkp
```

Then hit “Enter”

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Apr 26 15:45:45 2018
pi@raspberrypi:~ $ ls
99-fbturbo.conf  Documents  FreqShow_Small  Music        Public
Desktop          Downloads  logs             Pictures     python_games
pi@raspberrypi:~ $ sudo cp -r FreqShow_Small FreqShow_Small.bkp
```

If you typed this in correctly, type `ls` and you will see the new folder with `_bkp` on the end.

```
pi@raspberrypi:~$ ls
Desktop Downloads FreqShow_Small logs Music
pi@raspberrypi:~$ sudo cp -r FreqShow_Small FreqShow
pi@raspberrypi:~$ ls
99-fbturbo.conf Documents FreqShow_Small logs
Desktop Downloads FreqShow_Small_bkp Music
pi@raspberrypi:~$
```

Now that we have backed up our program to be safe, we need to view the file that needs to have the new values written to it.

I have a copy of the file for you to look at on the next page, find the values and where they are that you want to change, I have them highlighted in yellow, some of those you probably want to leave the same. Any way make sure you know ahead of time where to look for the values to change. It is always a good idea to not make too many changes at a time. If something goes wrong, its going to be much easier to find!

```

# FreqShow main application model/state.
# Author: Tony DiCola (tony@tonydicola.com)
#
# The MIT License (MIT)
#
# Copyright (c) 2014 Adafruit Industries
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.
# Enhancements over the original Freqshow by Dan Stixrud, WQ7T
import numpy as np
from scipy import signal
from scipy.fftpack import fft, rfft, fftshift

from rtlsdr import *

import freqshow

class FreqShowModel(object):
    def __init__(self, width, height):
        """Create main FreqShow application model. Must provide the width and
        height of the screen in pixels.
        """
        # Set properties that will be used by views.
        self.width = width
        self.height = height

        # Initialize auto scaling both min and max intensity (Y axis of plots).
        self.min_auto_scale = True
        self.max_auto_scale = True

        self.set_min_intensity(-10)
        self.set_max_intensity(50)

        # Initialize RTL-SDR library.
        self.sdr = RtlSdr()
        self.set_freq_correction(0) # (58ppm for unenhanced) can run test to determine this
        self.set_swap_iq(True)
        self.set_sample_rate(.230) # in MHz, must be within (.225001 <= sample_rate_mhz <= .300000) OR (.900001
        <= sample_rate_mhz <= 3.200000)
        self.set_zoom_fac(.05) # equal to the frequency span you want to display on the
        self.set_lo_offset(0.03) # Local Oscillator offset in MHz, slide the DC spike out of th
        self.set_center_freq(70.451500) # The 1st IF frequency of your radio
        self.set_gain('AUTO')
        self.set_fft_ave(3)
        self.set_tune_rate(.001) # in MHz
        self.set_sig_strength(0.00)
        self.set_kaiser_beta(8.6)
        self.set_peak(True) # Set true for peaks, set False for averaging.
        self.set_filter('nuttall') # set default windowing filter.

```

Once you have it very clear what you want to change and where it is, We will walk all the all the way through how, before you open it though.

When you first open the model.py file for editing it will look like this with a green square at the start of the program code on the first line.

```
GNU nano 2.2.6
█ FreqShow main application model/state.
# Author: Tony DiCola (tony@tonydicola.com)
#
# The MIT License (MIT)
#
# Copyright (c) 2014 Adafruit Industries
#
# Permission is hereby granted, free of charge, to any
# person obtaining a copy of this software and associated
# documentation files (the "Software"), to deal in the
# Software without restriction, including without
# limitation the rights to use, copy, modify, merge,
# publish, distribute, sublicense, and to permit
# others to do so, subject to the following conditions:
#
# The above copyright notice and this permission
# notice shall be included in all copies or
# substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY
# OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
# NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES
# OR OTHER LIABILITY, WHETHER IN AN ACTION OF
# CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
# OF OR IN CONNECTION WITH THE SOFTWARE OR THE
# RESULTS OF THE USE OF THE SOFTWARE.
# Enhancements over the original freqshow by Dan
import numpy as np
from scipy import signal
from scipy.fftpack import fft, rfft, fftshift

from rtlsdr import *
```

What we are going to do is use our “down arrow” key to navigate down to the line of text we want to edit, **you will not be using your mouse, or any other keys yet**. That green square tells you where you are.

```

        # Initialize auto scaling both min and max intensity
        self.min_auto_scale = True
        self.max_auto_scale = True

        self.set_min_intensity(-6)
        self.set_max_intensity(54)

        # Initialize RTL-SDR library.
        self.sdr = RtlSdr()
        self.set_swap_iq(True)
        self.set_sample_rate(.230) # in MHz, must be
        self.set_zoom_fac(.02) # equal to the frequency
        self.set_lo_offset(0.02) # Local Oscillator offset
        self.set_center_freq(70.451500)
        self.set_gain('AUTO')
        self.set_fft_ave(3)
        self.set_tune_rate(.001) # in MHz
        self.set_freq_correction(58) # (58ppm for uncalibrated)
        self.set_sig_strength(0.00)
        self.set_kaiser_beta(8.6)
        self.set_peak(True) # Set true for peaks, set false for
        self.set_filter('nuttall') # set default windowing
```

Im just running you through what to expect before you actually do it. Now that I have the green square at the start of my line where I have the center frequency value, I use the right arrow key to move it over the right parentheses where it has the current value.

```

self.set_min_intensity(-6)
self.set_max_intensity(54)

# Initialize RTL-SDR library.
self.sdr = RtlSdr()
self.set_swap_iq(True)
self.set_sample_rate(.230) # in MHz, must be
self.set_zoom_fac(.02) # equal to the frequency
self.set_lo_offset(0.02) # Local Oscillator offset
self.set_center_freq(70.451500)
self.set_gain('AUTO')
self.set_fft_ave(3)
self.set_tune_rate(.001) # in MHz
self.set_freq_correction(58) # (58ppm for uncalibrated)
self.set_sig_strength(0.00)
self.set_kaiser_beta(8.6)
self.set_peak(True) # Set true for peaks, set false for amplitude
self.set_filter('nuttall') # set default window

```

Next use your “Backspace” key to delete the current value one character at time. It should look like below once the value is deleted.

```

self.min_auto_scale = True
self.max_auto_scale = True

self.set_min_intensity(-6)
self.set_max_intensity(54)

# Initialize RTL-SDR library.
self.sdr = RtlSdr()
self.set_swap_iq(True)
self.set_sample_rate(.230) # in MHz, must be
self.set_zoom_fac(.02) # equal to the frequency
self.set_lo_offset(0.02) # Local Oscillator offset
self.set_center_freq()
self.set_gain('AUTO')
self.set_fft_ave(3)
self.set_tune_rate(.001) # in MHz
self.set_freq_correction(58) # (58ppm for uncalibrated)
self.set_sig_strength(0.00)
self.set_kaiser_beta(8.6)
self.set_peak(True) # Set true for peaks, set false for amplitude
self.set_filter('nuttall') # set default window

```

Now type the frequency in MHz of your 1st IF, mine is 70.451500 so I will retype it. Then use the left arrow key to move the green square all the way back to the left as shown below.

```

self.set_min_intensity(-6)
self.set_max_intensity(54)

# Initialize RTL-SDR library.
self.sdr = RtlSdr()
self.set_swap_iq(True)
self.set_sample_rate(.230) # in MHz, must be with
self.set_zoom_fac(.02) # equal to the frequency
self.set_lo_offset(0.02) # Local Oscillator offset
self.set_center_freq(70.451500) # frequency in MHz
self.set_gain('AUTO')
self.set_fft_ave(3)
self.set_tune_rate(.001) # in MHz
self.set_freq_correction(58) # (58ppm for my unen
self.set_sig_strength(0.00)
self.set_kaiser_beta(8.6)
self.set_peak(True) # Set true for peaks, set Fa
self.set_filter('nuttall') # set default windowing

```

Next I am going to change the freq correction ppm value so I will down arrow to that line.

```

self.set_sample_rate(.230) # in MHz, must
self.set_zoom_fac(.02) # equal to the fre
self.set_lo_offset(0.02) # Local Oscillator
self.set_center_freq(70.451500) # frequency
self.set_gain('AUTO')
self.set_fft_ave(3)
self.set_tune_rate(.001) # in MHz
self.set_freq_correction(58) # (58ppm for
self.set_sig_strength(0.00)
self.set_kaiser_beta(8.6)
self.set_peak(True) # Set true for peaks,
self.set_filter('nuttall') # set default wi

```

Then I will right arrow over to the right most parentheses of the ppm value below.

```

self.set_zoom_fac(.02) # equal to the freq
self.set_lo_offset(0.02) # Local Oscillator
self.set_center_freq(70.451500) # frequency
self.set_gain('AUTO')
self.set_fft_ave(3)
self.set_tune_rate(.001) # in MHz
self.set_freq_correction(58) # (58ppm for
self.set_sig_strength(0.00)
self.set_kaiser_beta(8.6)
self.set_peak(True) # Set true for peaks,
self.set_filter('nuttall') # set default wi

```


Again I will use the “Backspace” key to delete the current value.

```
self.set_zoom_fac(.02) # equal to the fre
self.set_lo_offset(0.02) # Local Oscillator
self.set_center_freq(70.451500) # frequency
self.set_gain('AUTO')
self.set_fft_ave(3)
self.set_tune_rate(.001) # in MHz
self.set_freq_correction(0) # (58ppm for my
self.set_sig_strength(0.00)
self.set_kaiser_beta(8.6)
self.set_peak(True) # Set true for peaks,
self.set_filter('nuttall') # set default wi
```

Then I will type the new value in that I want for a default, in this case 0

```
self.set_zoom_fac(.02) # equal to the f
self.set_lo_offset(0.02) # Local Oscillat
self.set_center_freq(70.451500) # frequen
self.set_gain('AUTO')
self.set_fft_ave(3)
self.set_tune_rate(.001) # in MHz
self.set_freq_correction(0) # (58ppm for
self.set_sig_strength(0.00)
self.set_kaiser_beta(8.6)
self.set_peak(True) # Set true for peak
self.set_filter('nuttall') # set default
```

In just a minute you will open this file yourself and modify it with your values.

I am going to close this file and save it. Notice I haven’t messed with any spacing, if I did, the program will not work. All those lines start just where they are supposed to. That’s why I walked you through this first, you have to be very careful about what you do here. Any way I’m done with the new values, so I am going to leave that green square just where it is.


At this point I hold down “Ctrl key” and press “X” to Exit

```
self.min_intensity = 1
if self.max_auto_scale:
    self.max_intensity = 1
self.range = None

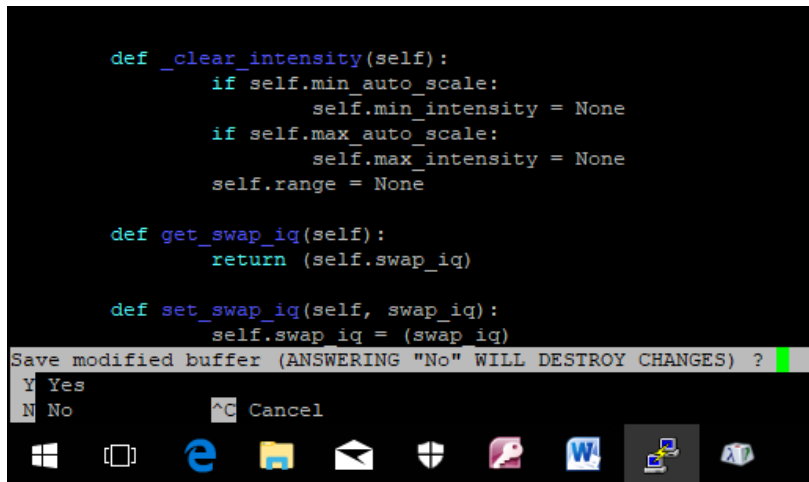
def get_swap_iq(self):
    return (self.swap_iq)

def set_swap_iq(self, swap_iq):
    self.swap_iq = (swap_iq)
```

^G Get Help ^O WriteOut
^X Exit ^J Justify



The Nano Editor will ask if I want to save the changes by hitting “y” or “n”
 There are times I know I accidentally messed things up so I will hit the “n” key, but in this case I’m sure I didn’t, so I am going to hit the “y” key.



```
def _clear_intensity(self):
    if self.min_auto_scale:
        self.min_intensity = None
    if self.max_auto_scale:
        self.max_intensity = None
    self.range = None

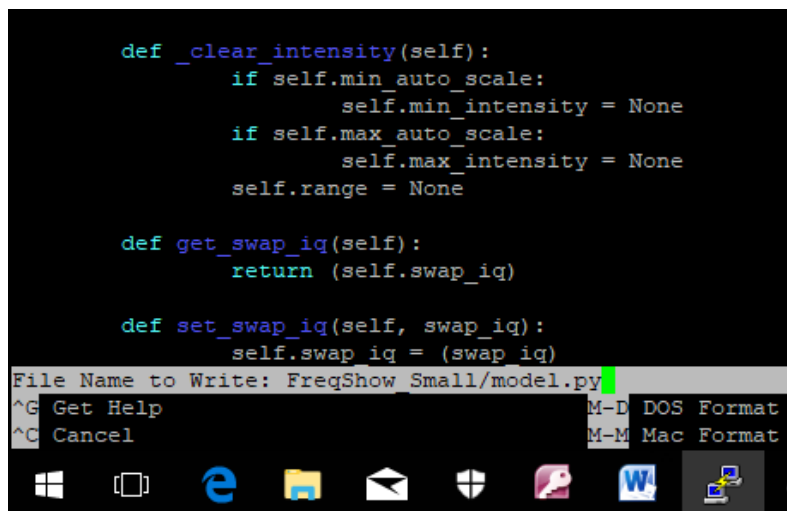
def get_swap_iq(self):
    return (self.swap_iq)

def set_swap_iq(self, swap_iq):
    self.swap_iq = (swap_iq)
```

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?

Y Yes
 N No ^C Cancel

Then it wants to confirm the overwrite so I hit “Enter” to the window below.



```
def _clear_intensity(self):
    if self.min_auto_scale:
        self.min_intensity = None
    if self.max_auto_scale:
        self.max_intensity = None
    self.range = None

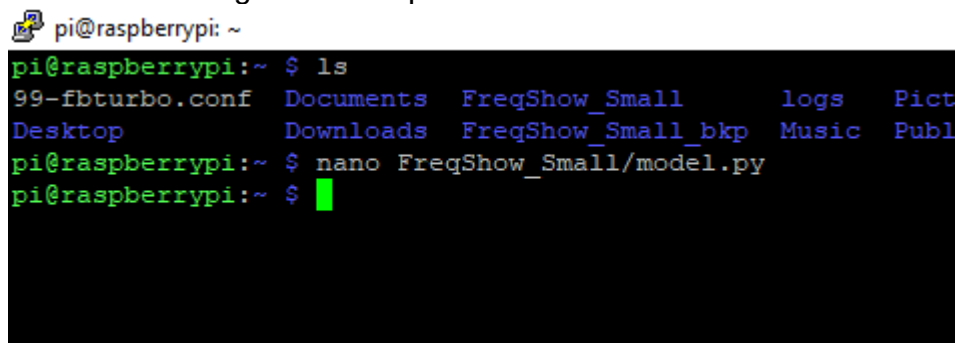
def get_swap_iq(self):
    return (self.swap_iq)

def set_swap_iq(self, swap_iq):
    self.swap_iq = (swap_iq)
```

File Name to Write: FreqShow_Small/model.py

^G Get Help M-D DOS Format
 ^C Cancel M-M Mac Format

This will return me to the command prompt and I can see the command I used to open the file for editing in the first place.



```
pi@raspberrypi: ~
pi@raspberrypi:~ $ ls
99-fbturbo.conf  Documents  FreqShow_Small  logs  Pict
Desktop          Downloads  FreqShow_Small_bkp  Music  Publ
pi@raspberrypi:~ $ nano FreqShow_Small/model.py
pi@raspberrypi:~ $
```

So there you go, to edit the file your self type

`nano FreqShow_Small/model.py`

OR

`nano FreqShow_Large/model.py`

depending again on your version of FreqShow.

follow the instructions previous to change the file with your new values and save it.

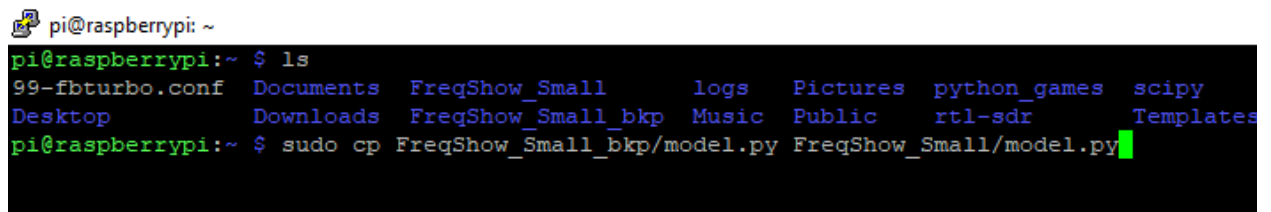
Once you have saved the file, stay in the SSH remote session and go start the FreqShow program again (not a reboot). If it comes back, check to make sure it is now using your default settings, not the original. Yeh! You were succesfull! You can skip the yellow highlighted step.

If the program trys, but doesn't start, something is wrong, so you can go back to your Putty terminal session, reopen the file and try to see what's wrong, or if you want to set the file back to what it was originally type:

`sudo cp FreqShow_Small_bkp/model.py FreqShow_Small/model.py`

OR

`sudo cp FreqShow_Large_bkp/model.py FreqShow_Large/model.py`

A terminal window screenshot showing a user at a Raspberry Pi. The prompt is 'pi@raspberrypi: ~'. The user has run 'ls' and the output shows various directories and files including '99-fbturbo.conf', 'Documents', 'FreqShow_Small', 'logs', 'Pictures', 'python_games', 'scipy', 'Desktop', 'Downloads', 'FreqShow_Small_bkp', 'Music', 'Public', 'rtl-sdr', and 'Templates'. The user then runs the command 'sudo cp FreqShow_Small_bkp/model.py FreqShow_Small/model.py' and the prompt returns.

```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ ls  
99-fbturbo.conf  Documents  FreqShow_Small    logs   Pictures  python_games  scipy  
Desktop          Downloads  FreqShow_Small_bkp Music  Public    rtl-sdr       Templates  
pi@raspberrypi:~ $ sudo cp FreqShow_Small_bkp/model.py FreqShow_Small/model.py
```

Test to see if FreqShow works again. Once you have FreqShow working again, you can type "exit" in the Putty terminal seesion and close that session.