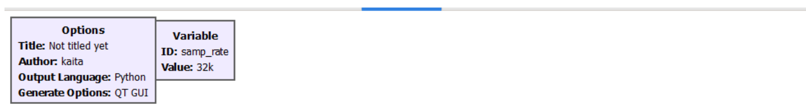
Taking Data with Gnu Radio Companion

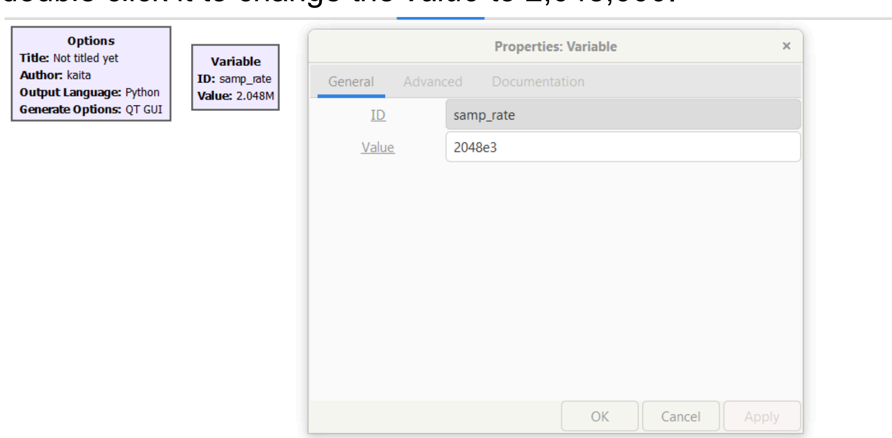
Flow Graph

Now that you have gnu radio on your computer it's time to start setting up your flow graph.
Here are some tutorials on how gnuradio works and how to get started https://wiki.gnuradio.org/index.php/Tutorials .

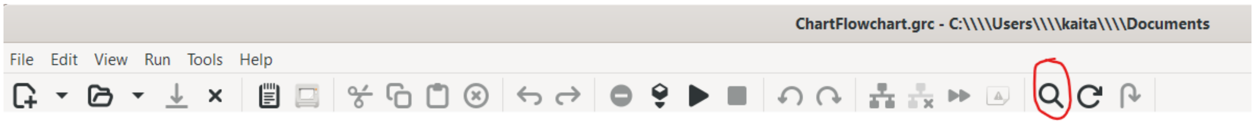
So we start off with a blank page.



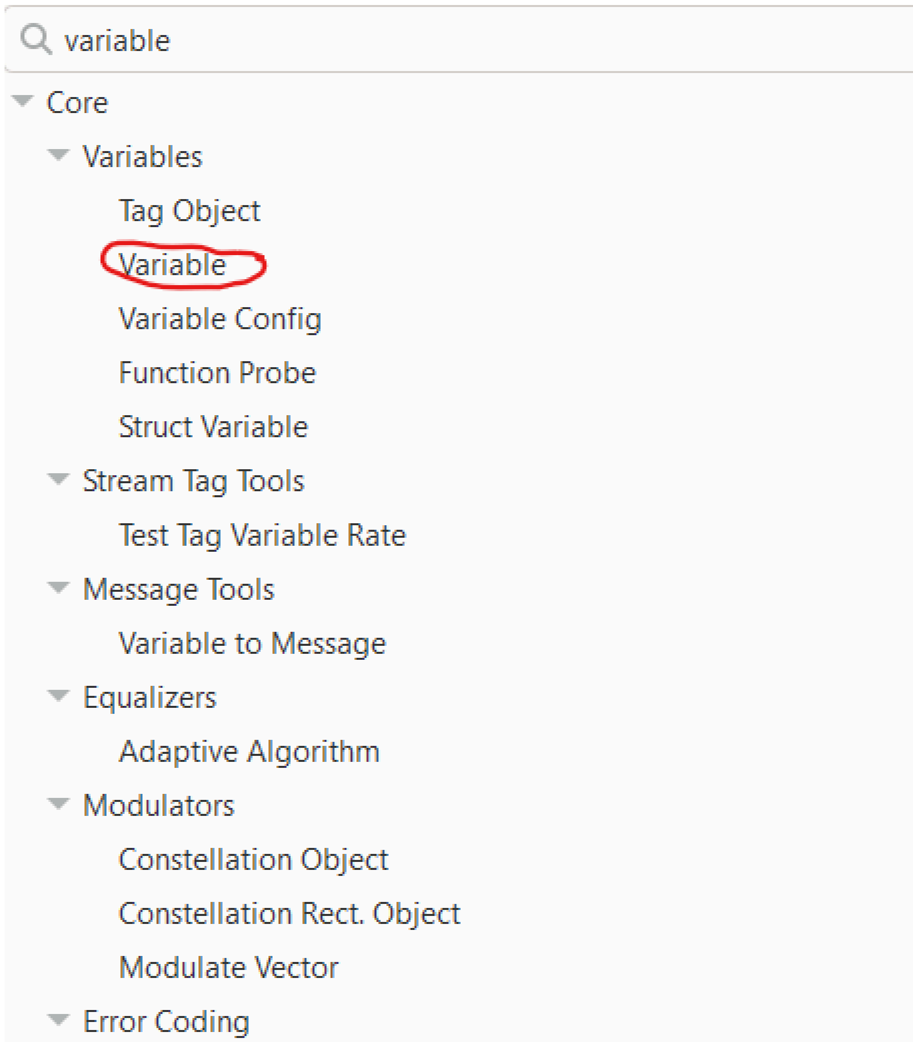
First step is to change the **samp_rate** and add in another variable that we will name **frequency**.
Samp_rate will already be on the page. The sample rate is determined by the SDR-RTL you have, mine had a sample rate of 2,048,000, it is best to look it up. On the block double click it to change the *value* to 2,048,000.



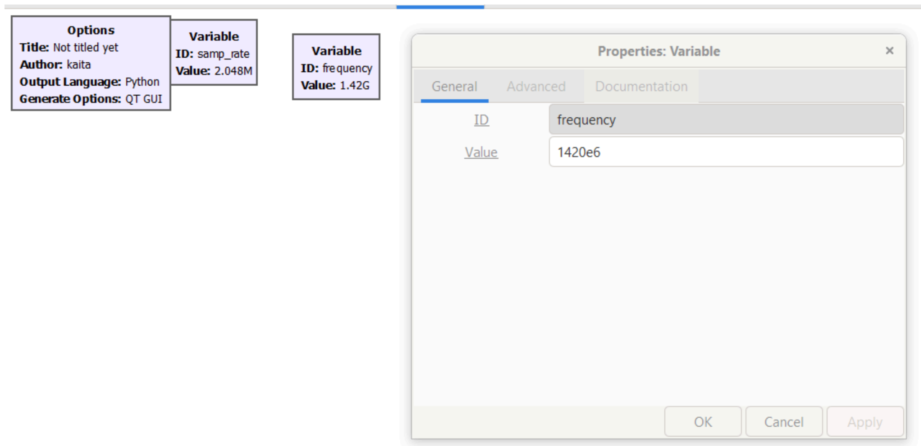
On the top of the page there is a magnifying class click on it, it will be used for the rest of the blocks.



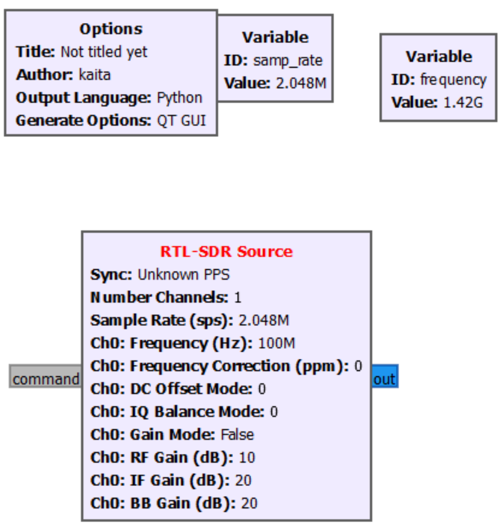
When you click on the magnifying glass you should see the search bar popping up on the right side of the page. Type in variable and you should see this list pop up.



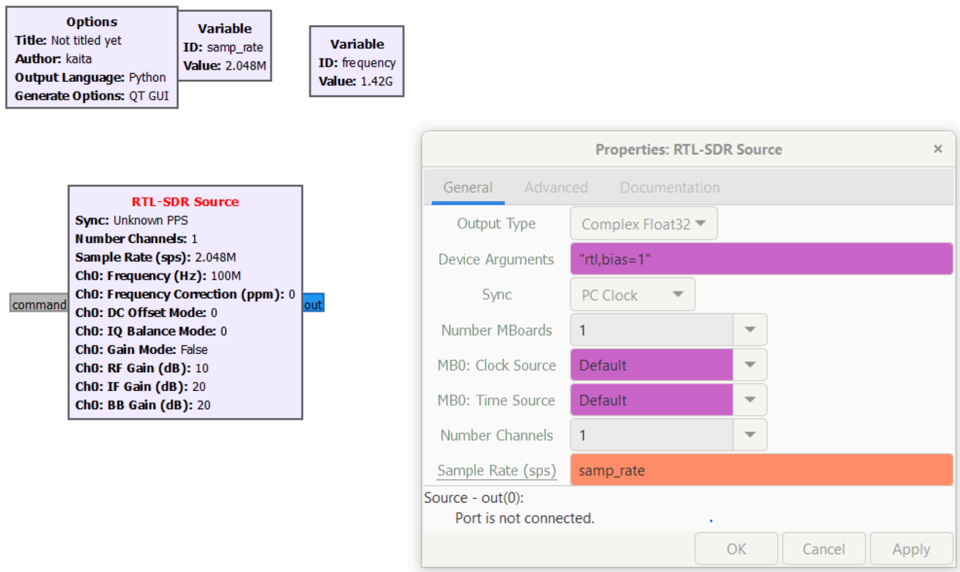
You can click and drag the variable into the work space. Once the block has appeared, double click and change the title from **variable_0** to **frequency** and then set the value to 1,420,000,000. This is the frequency of the galaxy that we want to be measuring.

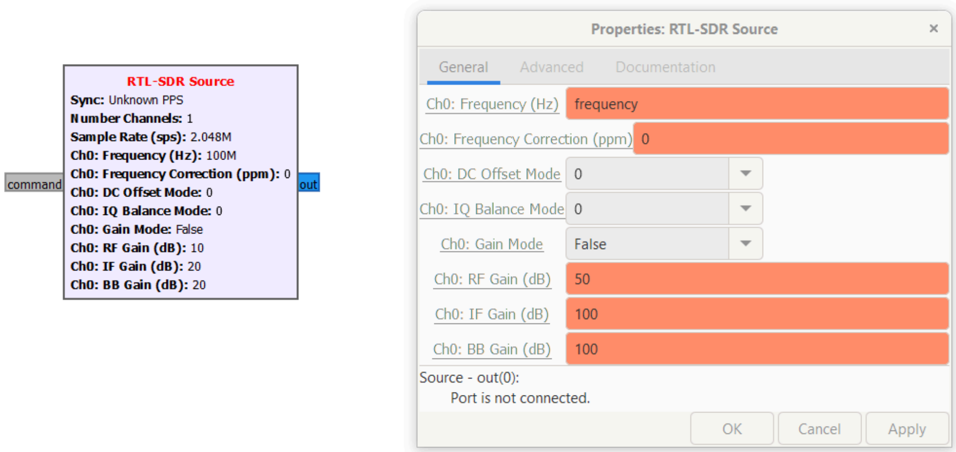


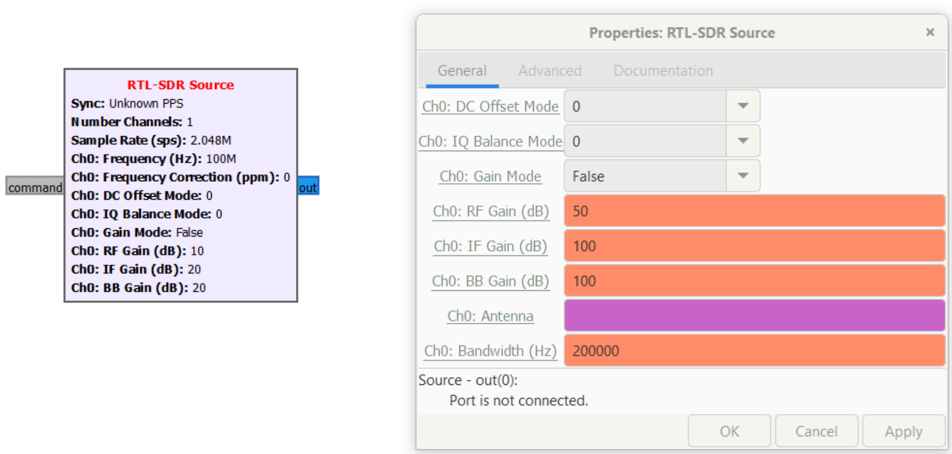
Now that the variables are set to the correct values, we want to bring in the **RTL-SDR Source**. This block can be found the same way that the variable block was. This block allows us to connect to the SDR we have plugged into our computer.



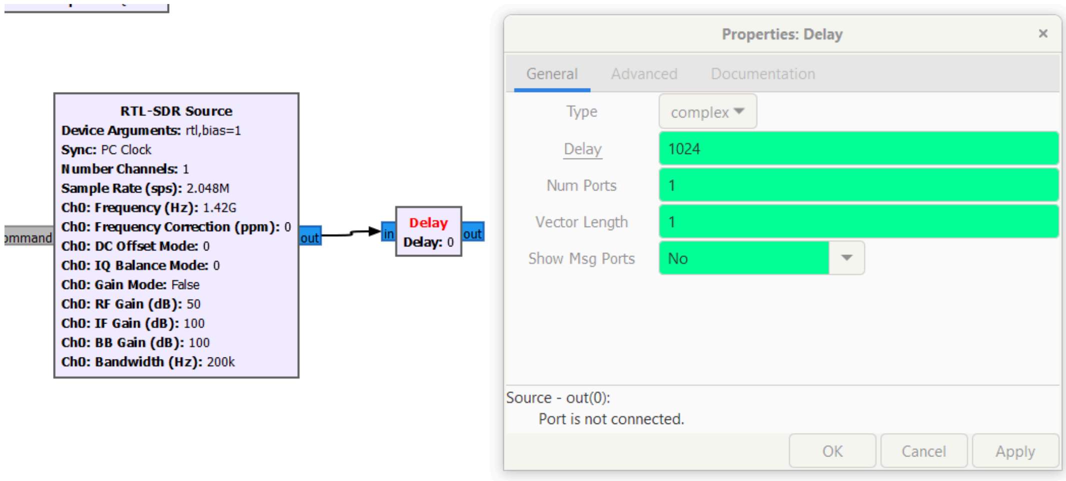
We are now going to change some of its settings.
First where it says device arguments in the settings you want to add in rtl,bais=1. This turns on the low noise amplifier attached to the horn at the start of the data taking section. Then you want to set sync to PC clock so that you have a way of taking time. Next in Ch0: Frequency (Hz) we are going to write **frequency** so that the device knows what frequency it is looking for. After that we change the RF, IF, and BB gain. The RF gain becomes 50 while the If and BB gain become 100, this allows us to pick up on the signal that is very faint. Last we change the bandwidth, we want the bandwidth to cover a bit of area so we change it from 0 to 200,000



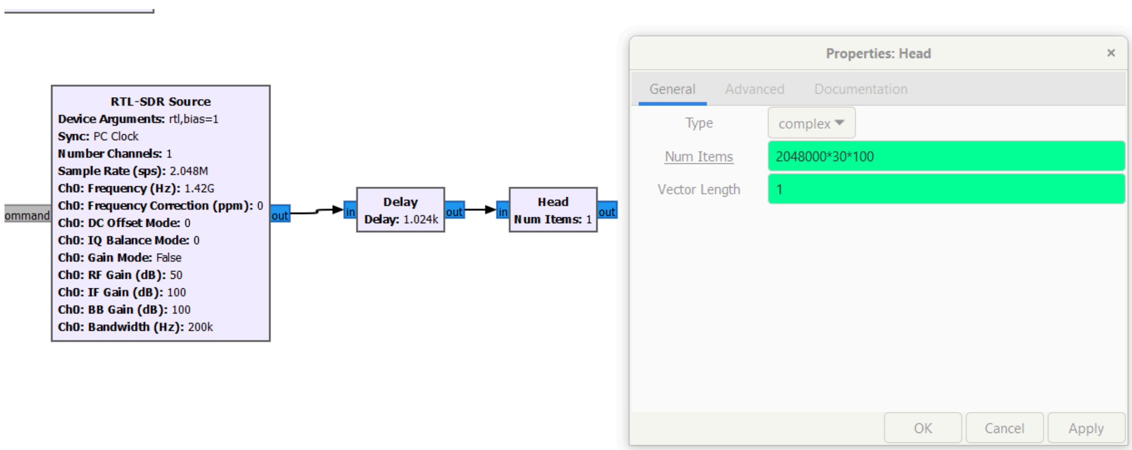




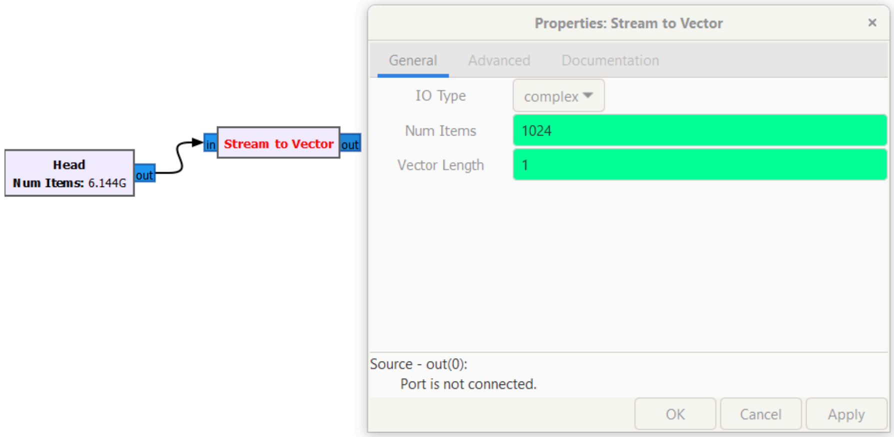
Now we want to add in a **Delay**. This is going to allow data to stack up and have a clearer trend than without it. The delay variable needs to be set to 1024, 1024 is a standard vector size when processing data so we have to be consistent through the whole flow chart. Connect the **RTL-SDR source** block to the **Delay** block.



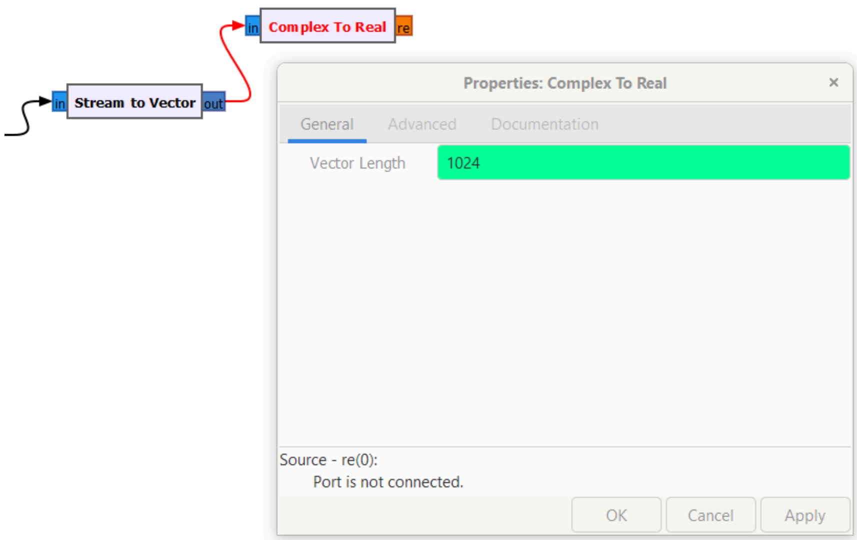
Next we are going to add a block called **Head**, this limits the amount of data points that are coming in so that you don't have an infinite amount of points.The way I figured out the amount of *Num. Items*/data points I wanted was by first looking at the **Samp_rate**, which is the amount of data point that are recorded in a second, and multiplying that by 30, 30 is the number of seconds I want to record. Then because I wanted some wiggle room with the amount of data I wanted, just in case something happed, I multiplied it all by 100. So what I inputted into *Num Items* was 2048e3*30*100. Then connect the **Head** block to the **Delay**.



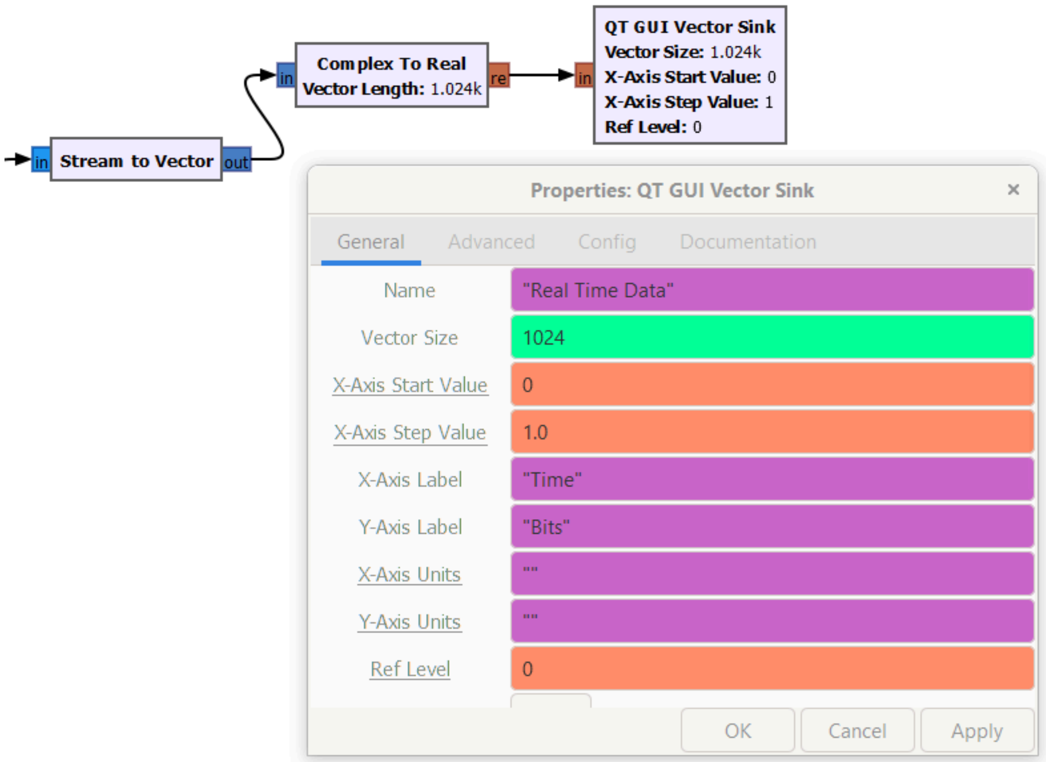
Now we want to add a **Stream to Vector** block. This takes the stream of data that is coming out of the SDR/Delay/Head and turns it into a vector. It is important to change it into a vector because the next few blocks only process vector data. In the **Stream to Vector** block you are going to want to change the Num Items from 2 to 1024. Attach to the **Head** block.

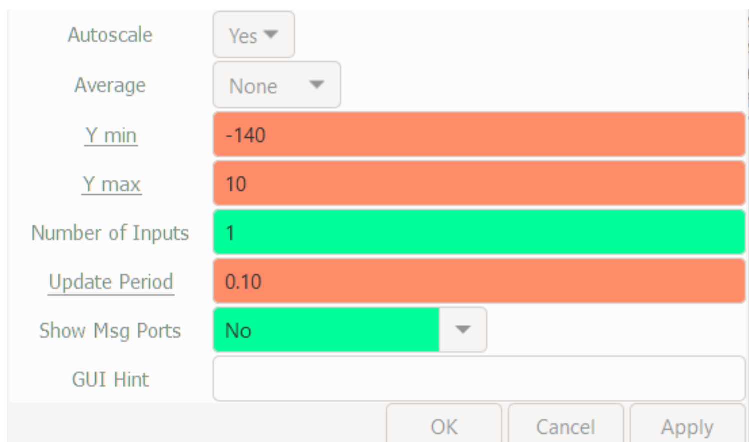


This is where the graph splits into the first graph and the continued signal processing. On the one branch we start with the block **Complex to Real**. This takes the imaginary numbers of the data and converts them to real numbers. In the block we have to set the vector length to 1024, like the **Stream to Vector**. Attach the **Complex to Real** block to the **Stream to Vector** block.

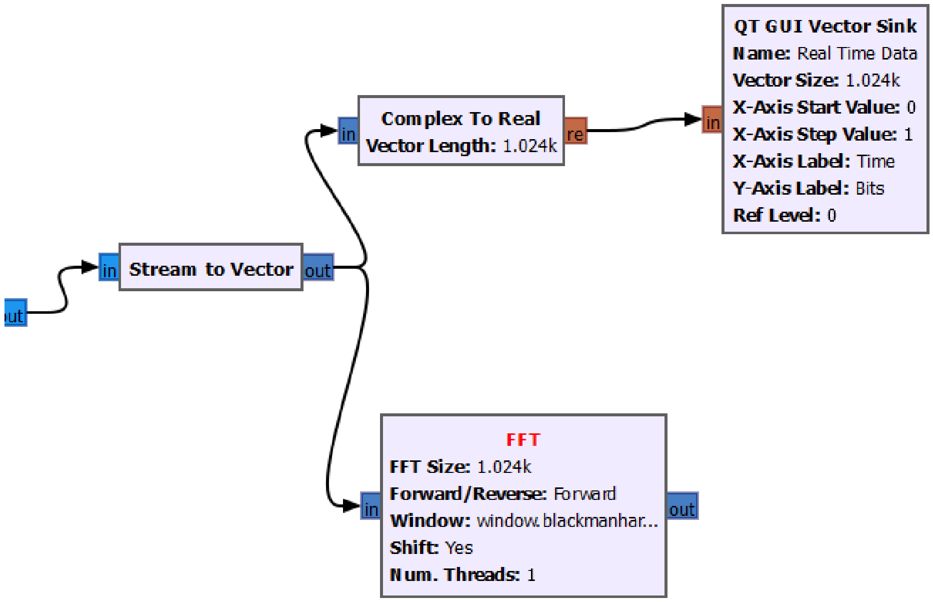


To the **Complex to Real** block we want to attach a **QT GUI Vector Sink** block, this block is going to show us the real time data in a graph. It will let us know what the data is doing and if there is a frequency or not. We need to change variables in this block to allow us to get a graph that makes sense, like titling and specifying what each axis is. The name of the graph can be anything you want but I called mine “Real time Data”, the x-axis label is time, and the y axis label is bits. We also want to change the graph from no autoscale to yes autoscale so that we can see the full range of data.

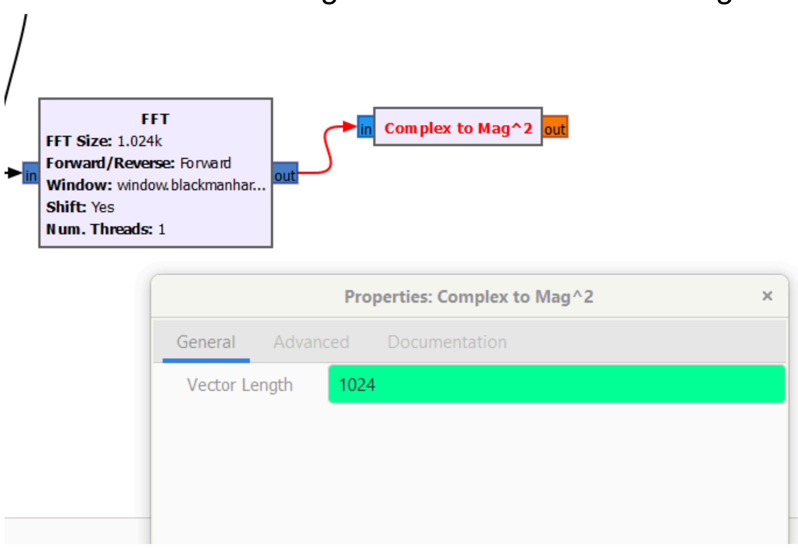




Now we go back to the **Stream to Vector**. We are going to attach an **FFT** block to the **Stream to Vector** block. FFT is short for Fast Fourier Transform, what this block does is takes all the data and transforms the data from the time base data, that the **QT GUI Vector Sink** is in, and turns it into a frequency based data. This block makes the whole spectrometer graph work. You don't have to change this block, just make sure that the block is set to Forward and the shift is set to yes.



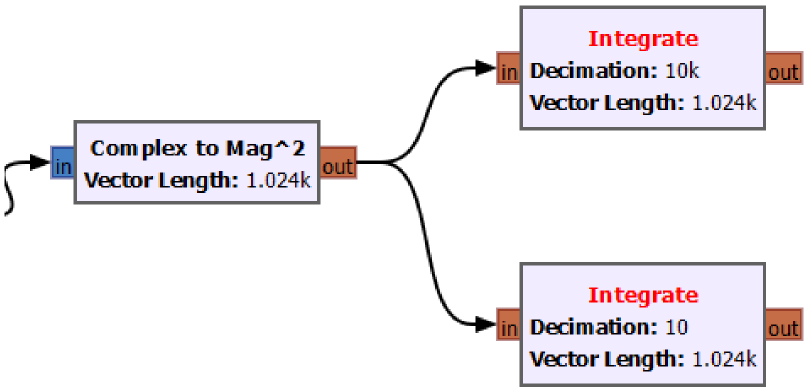
Complex to Mag^2 block is after the **FFT** block. This turns the data from imaginary to real usable graph data, like with **Complex to Real**, except it does this by squaring itself which is what gives it the magnitude. Make sure that the vector length is set to 1024 because that is the length of the vector data coming out of the FFT block.



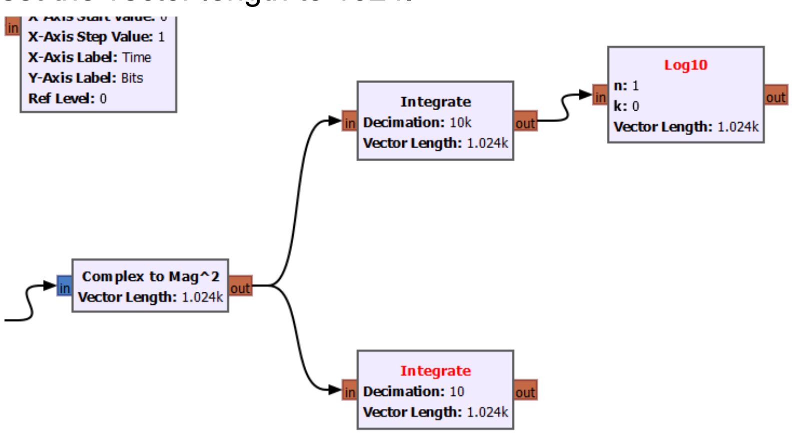
Next you are going to add two **Integration** blocks after the **Complex to Mag^2**. Two blocks of integration are needed because the **Waterfall Plot** and the **Vector Sink**/**File Sink**/**File Save** need to have different integration decimation than the other.

For the **Integration** block that is going to lead to the **File Sink**/**Vector Sink**/**File Save** you want to have the decimation set to 10,000 and the vector length set to 1024.

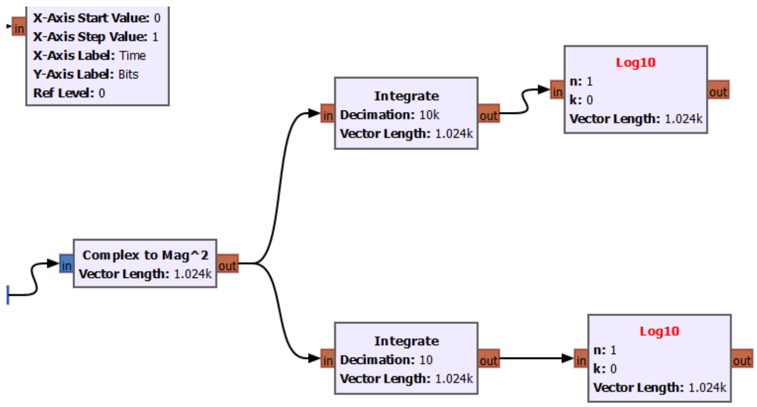
The **Integration** block that is going to lead to the waterfall plot needs the decimation to be set to 10 and the vector length needs to be set to 1024. For both of them you want to set them to the data type float.



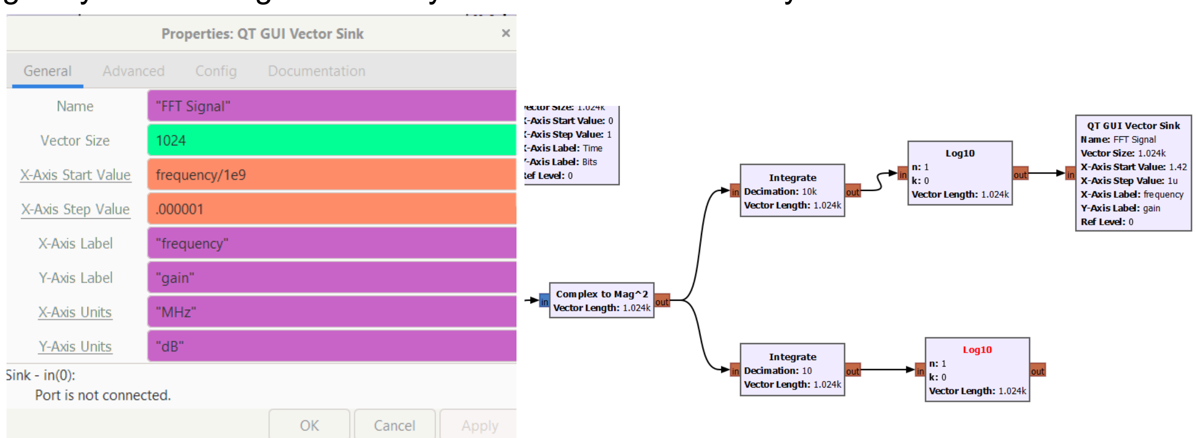
For the 10,000 **Integrate** block you need to attach to **Log10**. This block allows the graph to not be skewed towards large values. All you need to change on this block is set the vector length to 1024.



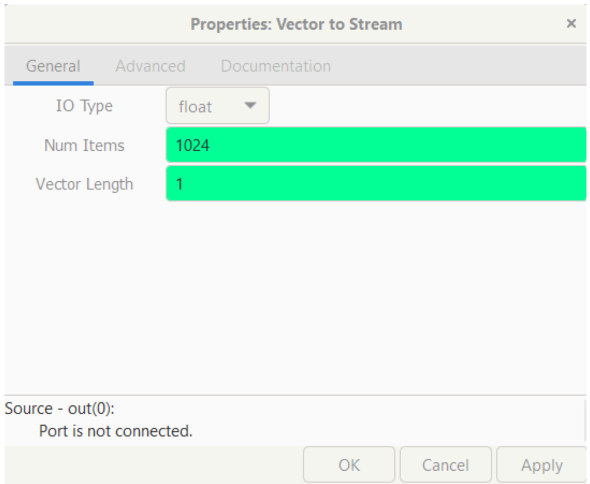
Now to the **Integrate** Block with the decimation of 10 we also want to attach the **Log10** with the vector length of 1024 to make the waterfall plot.

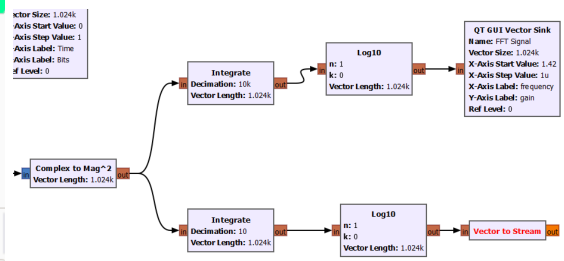


To the **Log10** that is attached to the Integrate decimation 10,000 you need to attach a **Vector Sink**. The **Vector Sink** will be your live spectrometer and show you where the galaxy is. Set the grid lines to yes and the autoscale to yes as well.

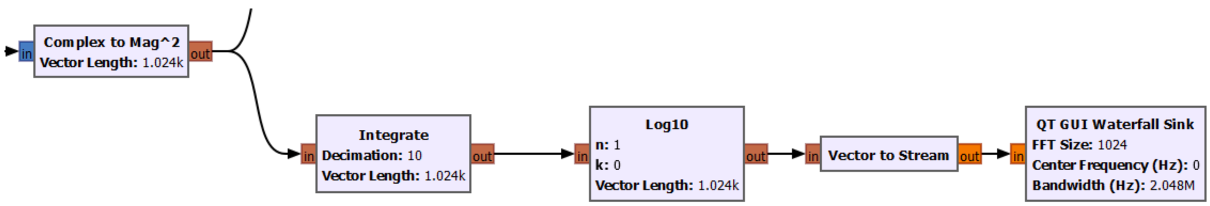


To the **Integrate** decimation 10 block you are now attaching the **Vector to Stream** block to the **Log10**. This allows you to make a continuous waterfall plot, which requires a float input. Set the num Items to 1024. Set the data type to float.



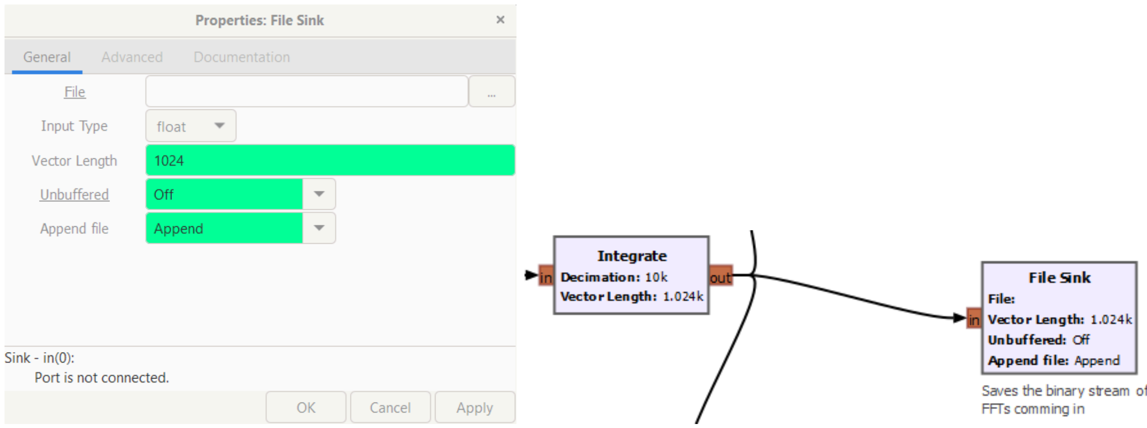


Then to the vector to stream you are attaching a **QT GUI Waterfall Sink**. All you are changing in this block is the data type from complex to float.

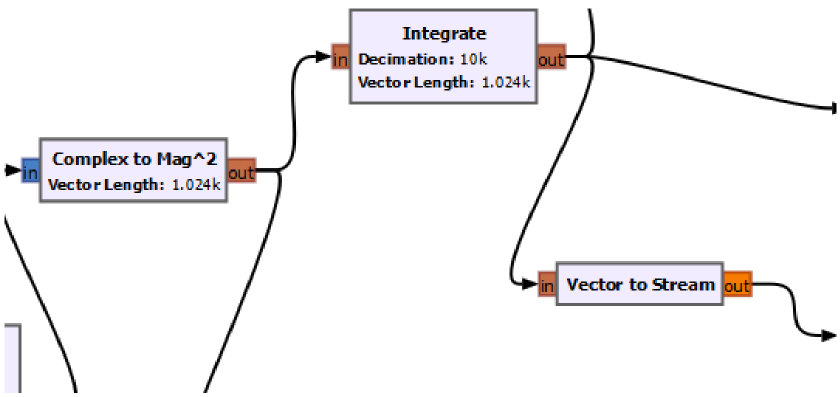


Now we are going to add in the blocks that will let us save the data we need to our computer. There are two blocks you need and one of them is custom made using a python block. These two blocks will connect to the **Integration** block with 10k.

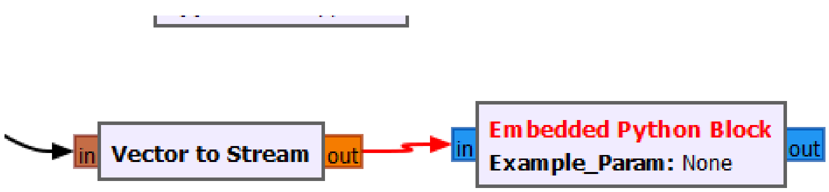
Drag File Sink from the side bar. This block is just going to save all the data coming in. You have to set where you want it to save on your computer in the file line, you also have to name it in the file name as well. The vector length needs to be set to 1024 and the *append file* needs to be set to append. This is then connected to the **Integration** block of 10k. It will save as binary code with nothing like frequency or any other informative data coming in. That is where the custom file save block comes in.



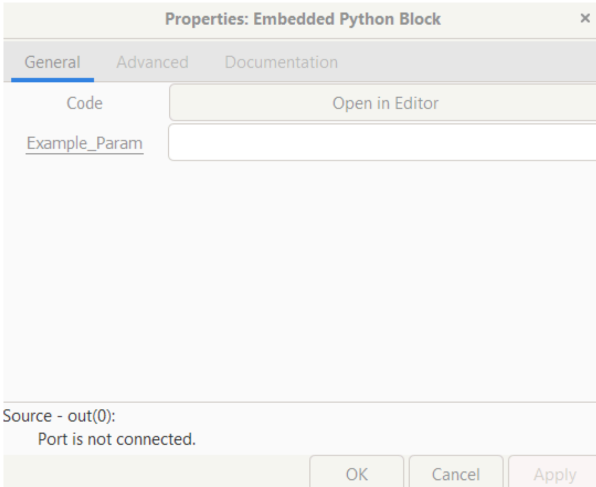
First before you get to the custom file save you need to add another **Vector to Stream** with num Items of 1024 and data type of float.

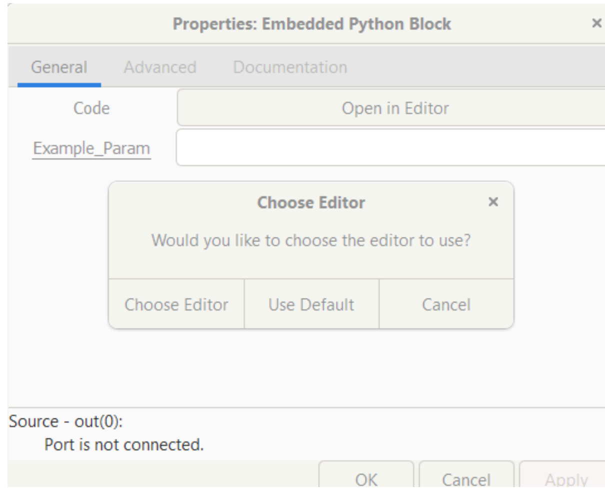


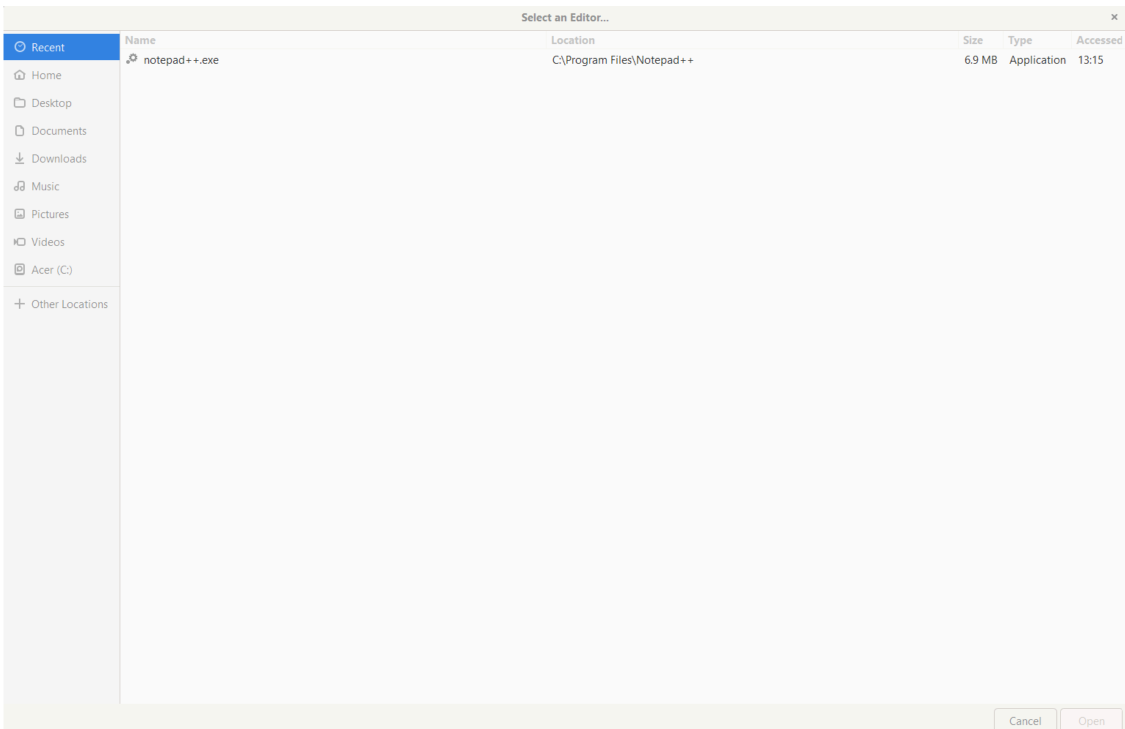
Now pull out an **Embedded Python Block** and connect it to the **Vector to Stream** block.



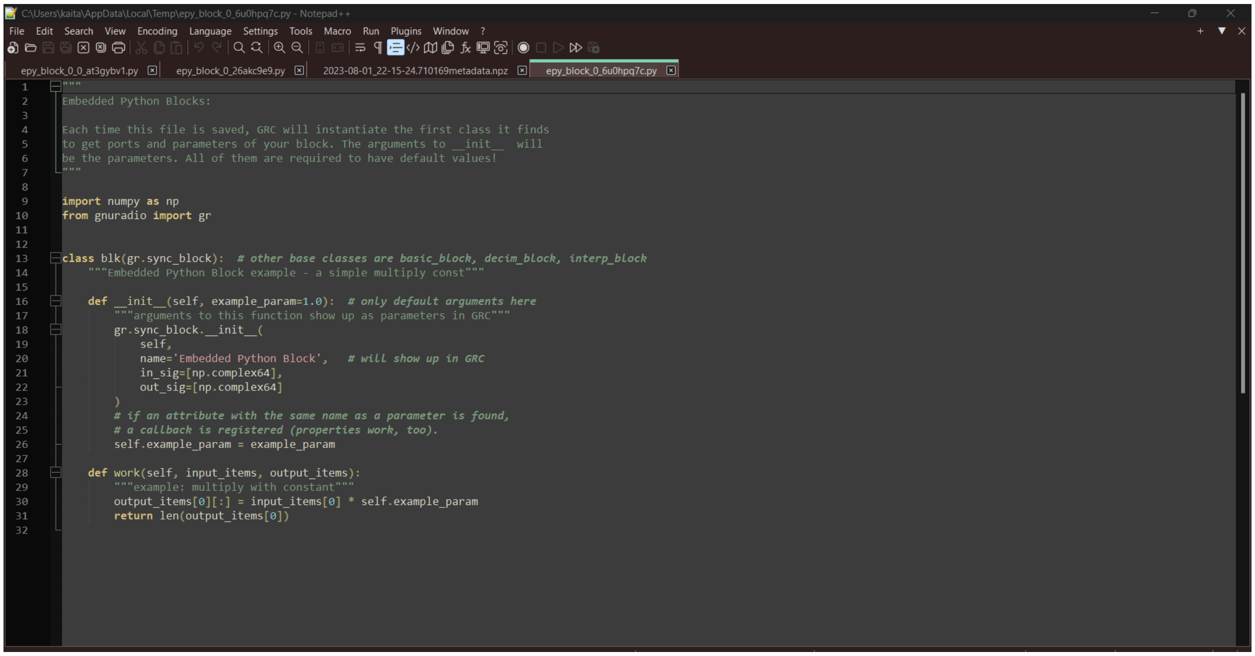
Then you are going to open the Python block in your chosen editor, mine is notepad++ or if you are on apple/linux you may be able to click the default editor.







Once you have selected the editor and have opened it. You should land on this page.




```python
"""
Embedded Python Blocks:

Each time this file is saved, GRC will instantiate the first class it finds
to get ports and parameters of your block. The arguments to __init__  will
be the parameters. All of them are required to have default values!
"""

import numpy as np
from gnuradio import gr


class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

    def __init__(self, example_param=1.0):  # only default arguments here
        """arguments to this function show up as parameters in GRC"""
        gr.sync_block.__init__(
            self,
            name='Embedded Python Block',   # will show up in GRC
            in_sig=[np.complex64],
            out_sig=[np.complex64]
        )
        # if an attribute with the same name as a parameter is found,
        # a callback is registered (properties work, too).
        self.example_param = example_param

    def work(self, input_items, output_items):
        """example: multiply with constant"""
        output_items[0][:] = input_items[0] * self.example_param
        return len(output_items[0])

```

You are going to delete everything.
Here is the code you want to have to save the file.

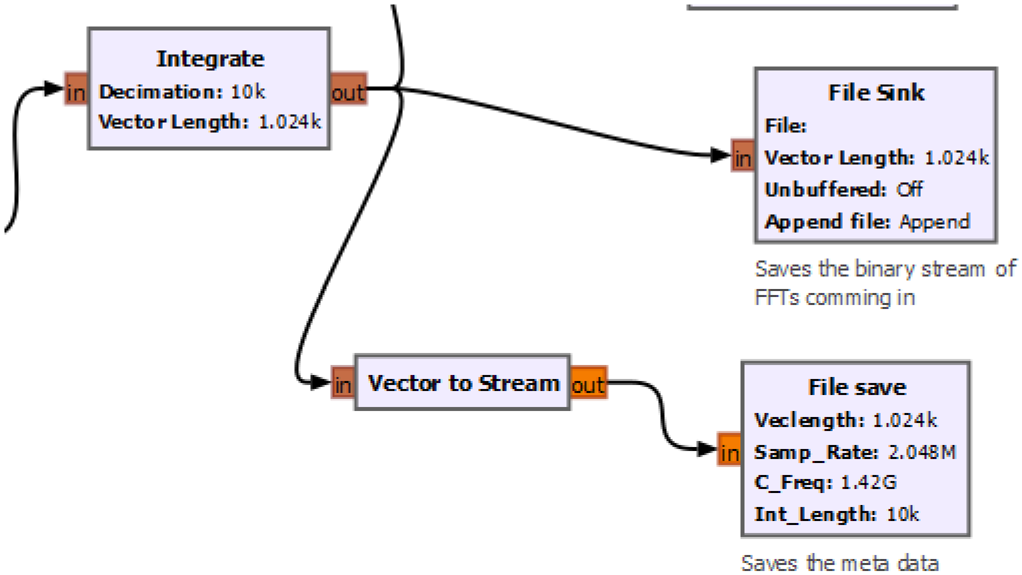
```python
import numpy as np
import time
import datetime
import os
from gnuradio import gr


class blk(gr.sync_block):

    def __init__(self, veclength=1024,samp_rate=2048e3, c_freq=1.420e6,
int_length=100):
        self.veclength=veclength
        self.samp_rate=samp_rate
        self.c_freq=c_freq
        self.int_length=int_length
        self.times = []
        self.start_time = None
        self.data_file = None
        gr.sync_block.__init__(self,
            name = "File save",
            in_sig = [np.float32],
```

```
            out_sig = None)


    def work(self, input_items, output_items):
        self.times.append(time.time())
        if self.start_time is None:
           self.start_time = time.time()

        if self.data_file is None:
            filebase = str(datetime.datetime.now()).replace(' ', '_')
            filebase = filebase.replace(':', '-')
        self.data_file = os.path.join(r'C:\Users\kaita\Documents', filebase + 'metadata.npz')
        if self.data_file is not None:
               np.savez(self.data_file, data=datetime.date.today(), start_time=self.start_time,
end_time=time.time(), samp_rate=self.samp_rate, frequency=self.c_freq,
                        vector_length=self.veclength, int_length=self.int_length,
data_file=self.data_file, times=self.get_times(), dtype=[np.float32], allow_pickle=True)

        return len(input_items)

    def get_times(self):
        return self.times
```

Save to program and it should look something like this


```
import numpy as np
import time
import datetime
import os
from gnuradio import gr


class blk(gr.sync_block):

    def __init__(self, veclength=1024,samp_rate=2048e3, c_freq=1.420e6, int_length=100):
        self.veclength=veclength
        self.samp_rate=samp_rate
        self.c_freq=c_freq
        self.int_length=int_length
        self.times = []
        self.start_time = None
        self.data_file = None
        gr.sync_block.__init__(self,
            name = "File save",
            in_sig = [np.float32],
            out_sig = None)


    def work(self, input_items, output_items):
        self.times.append(time.time())
        if self.start_time is None:
           self.start_time = time.time()

        if self.data_file is None:
            filebase = str(datetime.datetime.now()).replace(' ', '_')
            filebase = filebase.replace(':', '-')
        self.data_file = os.path.join(r'C:\Users\kaita\Documents', filebase + 'metadata.npz')
        if self.data_file is not None:
               np.savez(self.data_file, data=datetime.date.today(), start_time=self.start_time, end_time=time.time(), samp_rate=self.samp_rate, frequency=self.c_freq,
                        vector_length=self.veclength, int_length=self.int_length, data_file=self.data_file, times=self.get_times(), dtype=[np.float32], allow_pickle=True)

        return len(input_items)

    def get_times(self):
```

You may need to reconnect your block to the rest of the flow chart but after you do that it should look something like this.



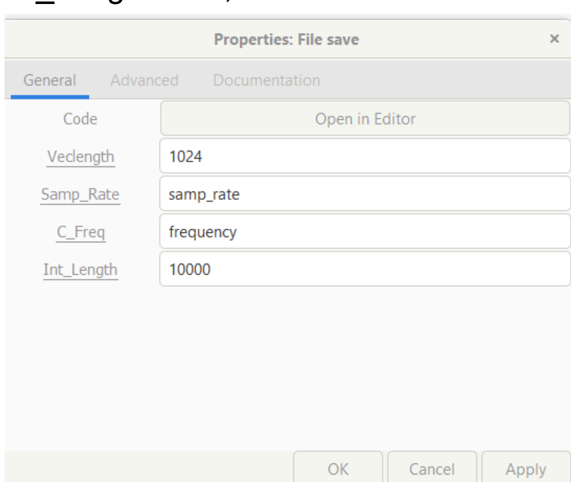
Your embedded python block is now a File save Block. Now you need to set the Veclength, Samp_rate, C_Freq, and the Int_Length.

Veclength = 1024

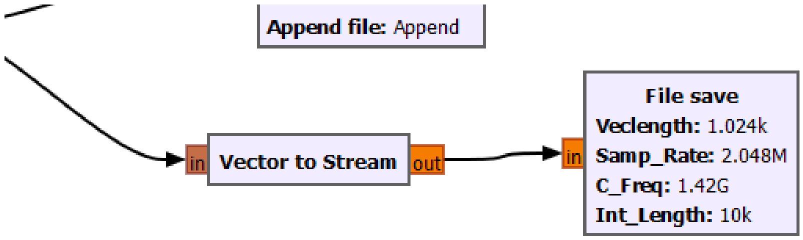
Samp_rate = samp_rate

C_Freq = frequency

Int_Length = 10,000



Now your flow graph should look like this.



Now you have a perfect flow graph for picking up the 21 cm. line of the galaxy.
The graph in full.