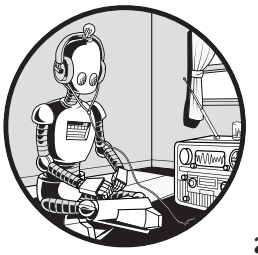
4

CREATING AN AM RECEIVER



In this chapter, we'll dive right into building your first software-defined radio: an amplitude-modulated (AM) receiver. Like an AM car radio, it will be able to take in AM radio signals and convert them to listenable audio. Rather than work with live radio signals, however, we'll test out the radio with a file containing captured radio data. This way you won't have to worry about any hardware just yet.

At this early stage of your SDR journey, there are two main things you need to learn: how to use GNU Radio Companion and the theory behind how SDRs are built. This chapter will focus on the former. As such, for now, we'll mostly gloss over the radio theory behind the receiver we build. Without this theory, some of the steps you take may not make sense yet, and it may feel like you're following a rote set of instructions, almost like building a model airplane. Rest assured, though: the intention is not to give you

simple cookbook instructions and send you on your way. We'll keep coming back to this project in the next few chapters to dig into the details of how the AM receiver, and radios in general, work.

Open up GNU Radio Companion and let's get started!

Setting Up the Variables and Entries

Create a new flowgraph by clicking **File ▸ New ▸ QT GUI** or by pressing CTRL-N. This will bring up a mostly empty starting flowgraph with just two blocks already in the workspace. Every new flowgraph starts with both of these blocks. The first is the `Options` block, which contains some basic documentation and settings for the flowgraph. The other is a `Variable` block, representing the variable called `samp_rate`. In general, you'll employ these `Variable` blocks to store values used throughout the flowgraph, just like defining variables in text-based programming languages like C or Python. As you might guess, this particular variable has something to do with the sample rate for the flowgraph.

Although not strictly necessary, it's good practice to add some basic information to the `Options` block. Double-click the block and then change the Title to **`AM Receiver`** and the Description to **`My first AM radio receiver`**.

Next, add a `QT GUI Entry` block to the flowgraph, as shown in Figure 4-1. (Remember, you can find new blocks using CTRL-F and the search box.)

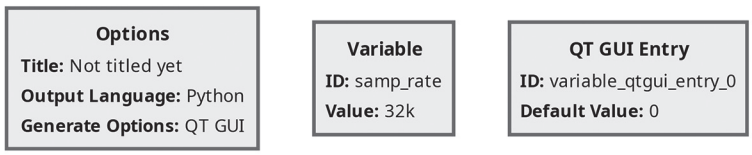


Figure 4-1: Adding a `QT GUI Entry`

Like a `Variable` block, a `QT GUI Entry` stores a value for use throughout the flowgraph. Unlike a `Variable` block, however, you can change a `QT GUI Entry` block's value in real time while the flowgraph is running. We tend to put our `Variable` and `QT GUI Entry` blocks together at the top of the flowgraph, but this isn't necessary. The flowgraph will function the same regardless of the position of the blocks in your workspace.

Double-click the new **`QT GUI Entry`** block to bring up its properties window. Change the ID to `freq` and the Default Value to `880e3`. This is exponential notation for 880,000. It's a common programming-language way of rendering 880×10^3. You can think of the value after the `e` as the number of zeros that GNU Radio Companion will add to the end of the number before the `e`. Exponential notation is especially useful for large numbers, making them much easier to read. For instance, `600000000` is not as clear at first glance as `600e6`. Remember this notation. You'll be using it a lot.

There's just one problem with what you've typed: if you click the Apply button, you'll see an error at the bottom telling you that the Default Value is invalid, as shown in Figure 4-2.

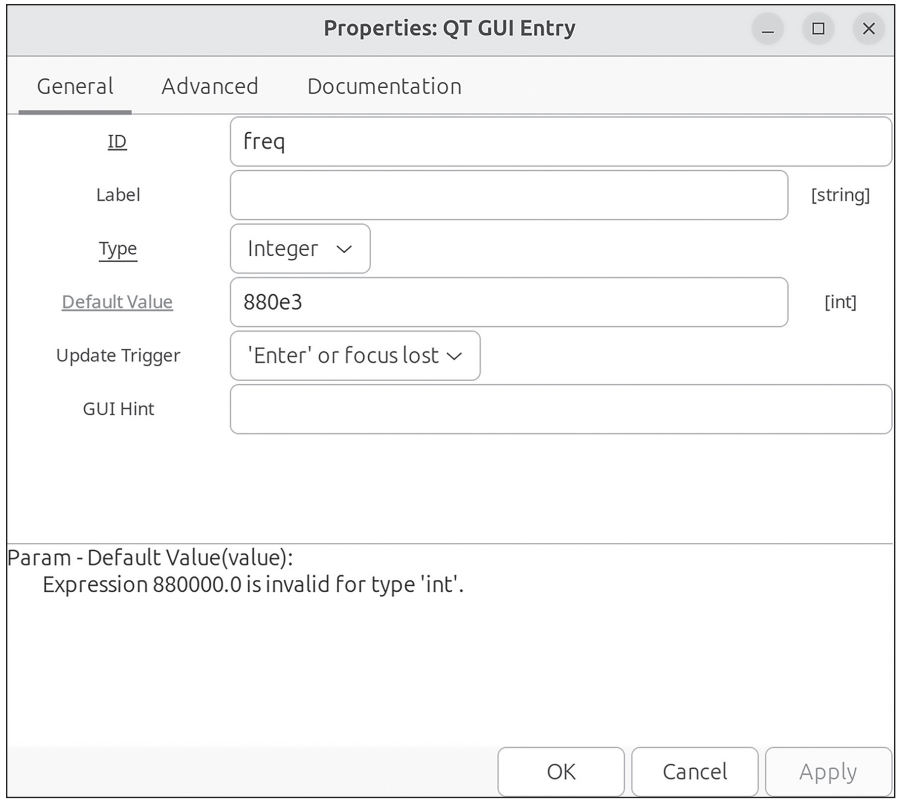


Figure 4-2: The `QT GUI Entry` error

The problem here is related to *data types*. You learned that flowgraphs can be thought of as numbers flowing out of sources, through blocks, and into sinks, but you haven't learned anything about what kind of numbers those are. Just as in programming languages like Python, Java, or C++, there are different types of data in GNU Radio Companion. As you can see on your screen (and in Figure 4-2), the Type property of the `QT GUI Entry` block is set to `Integer`, meaning it can accept positive or negative numbers without a decimal point, like `17` or `-1293`. It turns out, however, that exponential notation produces a floating point–typed value: a number with some digits after the decimal point, such as `3.14159` or `-8.9`. To make the Type of the `QT GUI Entry` compatible with the Default Value we've given it, click the pull-down menu containing `Integer` and instead select **`Float`**. Then click **Apply**, and the error should go away, as shown in Figure 4-3.

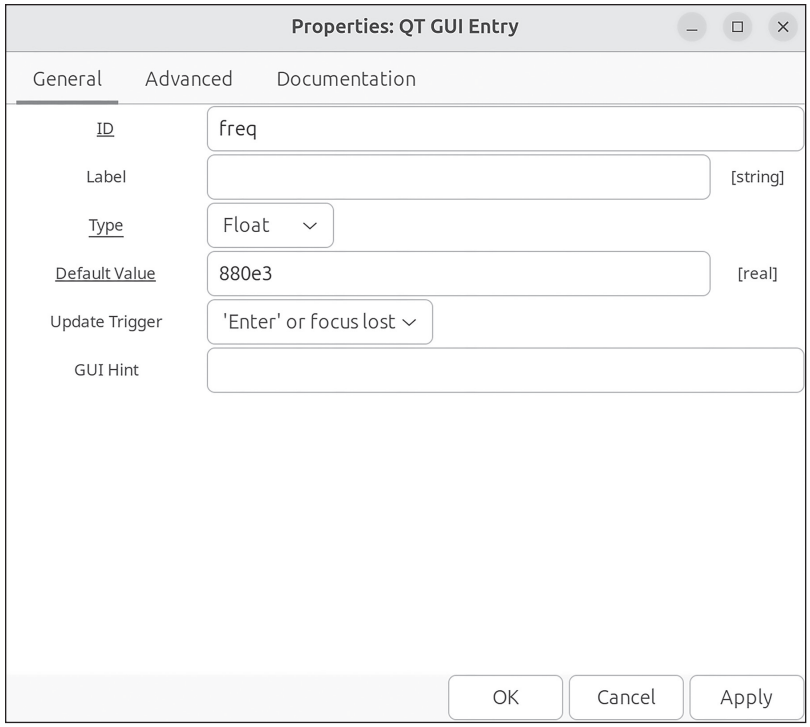


Figure 4-3: The corrected `QT GUI Entry` block

Click **OK** and note that the rendering of the block in the workspace changes to reflect the value you've entered. This is useful because even as our radio flowgraph gets progressively more complicated, you can see a lot about how it works from a top-level view, all at a glance.

Next, you'll add a second `QT GUI Entry` with a different ID and Default Value. Instead of adding it like before, though, you can simply copy and paste the first one by clicking the existing **`QT GUI Entry`** and pressing CTRL-C (or selecting **Edit ▸ Copy** from the menu bar), followed by CTRL-V (or **Edit ▸ Paste**). Notice that the new block appears with `freq_0` as its ID, as shown in Figure 4-4.

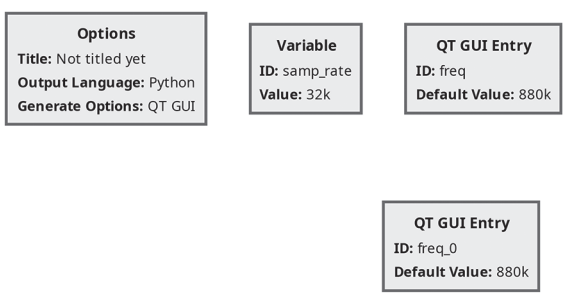


Figure 4-4: Copying the `QT GUI Entry` block

Double-click the new block and set its ID to `center_freq` with a Default Value of `900e3`. Because you copied the block, it already has its Type set to `Float`. When you're done, your flowgraph should look like Figure 4-5.

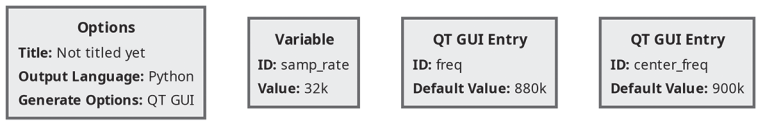


Figure 4-5: The second `QT GUI Entry`, configured

Notice that we've moved the new `QT GUI Entry` up to the top with the other blocks. Again, you may find it useful to keep all the `Variable` and `QT GUI Entry` blocks together at the top of the flowgraph.

Adding a Source of Radio Data

Next, we'll add a source block to bring radio data into the flowgraph. First, grab a `File Source` block and add it to the workspace. This will allow you to use a file as the radio data input to the flowgraph, so you won't require any SDR hardware. The file contains actual raw radio data that an enterprising SDR aficionado captured from the airwaves and stored for later use. If you were building a fully working radio, you'd use a different source block instead of the `File Source`. This alternative source would interface with your SDR hardware and provide real-time radio data to your flowgraph.

As before, double-click the newly placed block to set up its properties. In the File selection, click the three dots, then navigate to the location of the project files you downloaded earlier from *https://nostarch.com/practical-sdr*. Select the one named *ch_04/am_ broadcast_02_c900k_s400k.iq*. You'll see a warning in the lower portion of the properties window telling you that a port isn't connected, but you'll fix that in a moment, so don't worry about it. At this point your property window should look similar to Figure 4-6. Click **OK** to return to the workspace.

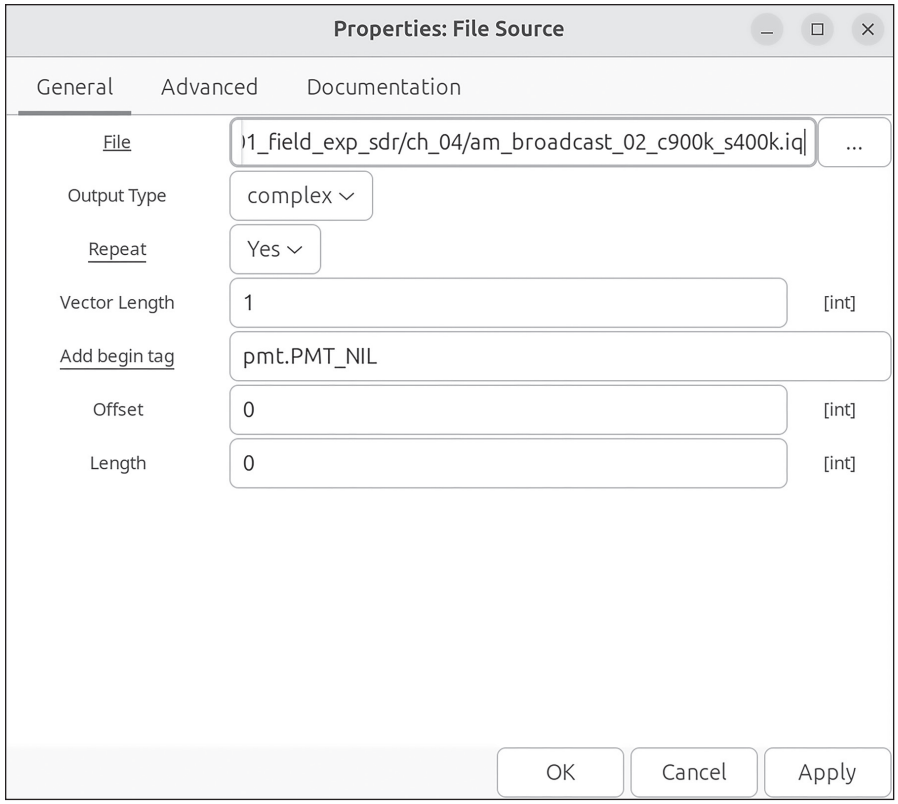


Figure 4-6: The `File Source` block properties

Now's a good time to save the project, so press CTRL-S or select **File ▸ Save**. We can give it any legal Linux filename, but let's use *first_am_rx.grc* (*rx* is shorthand for *receiver*). When working on your flowgraphs, remember to save early and often.

NOTE *From now on, we're going to assume you know how to search for blocks and add them to the design. We'll also assume you know how to bring up the properties listing for a block and change the relevant values. As such, we'll stop spelling out each step of these processes.*

Processing the Signals

The next several blocks we'll add will work together to process the radio signal from the `File Source`. Again, in this chapter we won't focus much on the details of what these blocks are doing or how exactly they work; we'll explore those questions later in the book. For now, our concern is building a working flowgraph.

First, place a `Signal Source` block into your workspace. You'll use this to generate an infinitely repeating sequence of values representing a sinusoid.

In the block's properties, you're going to do something a little different: for the Frequency property, you aren't going to enter a simple number. Instead, type `center_freq - freq`, as shown in Figure 4-7.

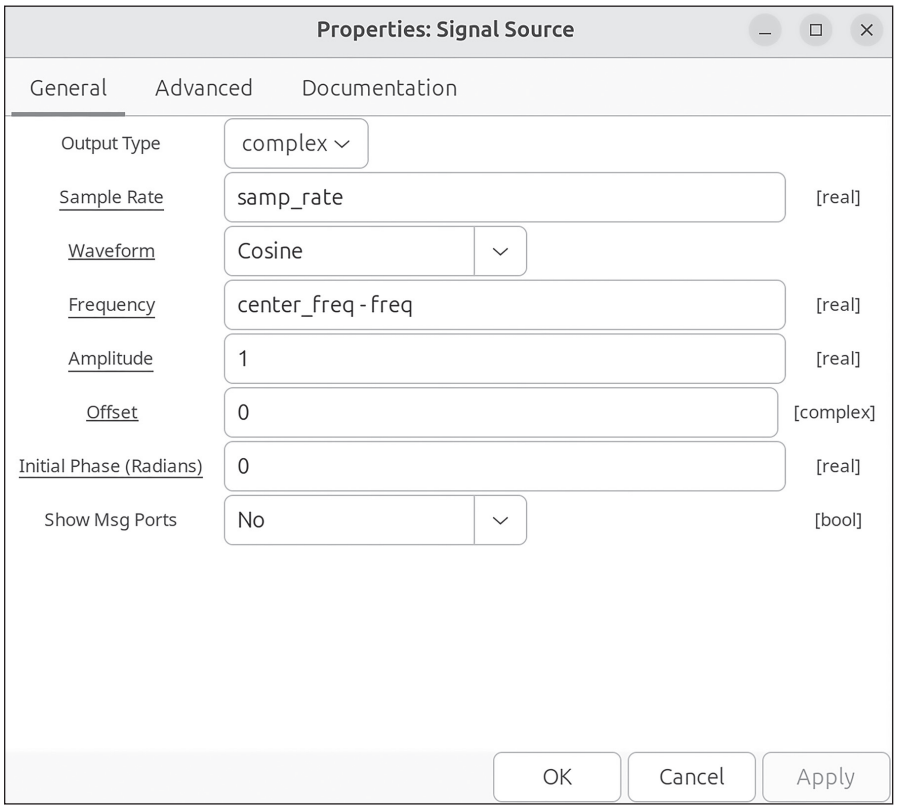


Figure 4-7: The `Signal Source` properties

Can you see what you just did here? Instead of entering a fixed number for a property, you can enter variables. And not just variables, but mathematical expressions involving multiple variables. In fact, as you'll see later, you can enter almost any legal Python expression as a property for a block and it will work. This will turn out to be very useful.

In this case, you're setting the frequency of the `Signal Source` using the two `QT GUI Entry` blocks you created. Specifically, you're subtracting the value of the block with ID `freq` from the value of the block with ID `center_freq`. Notice that when you click **OK** and go back to your workspace view, you don't see the math expression you just typed, but simply the number that results from it, as shown in Figure 4-8.

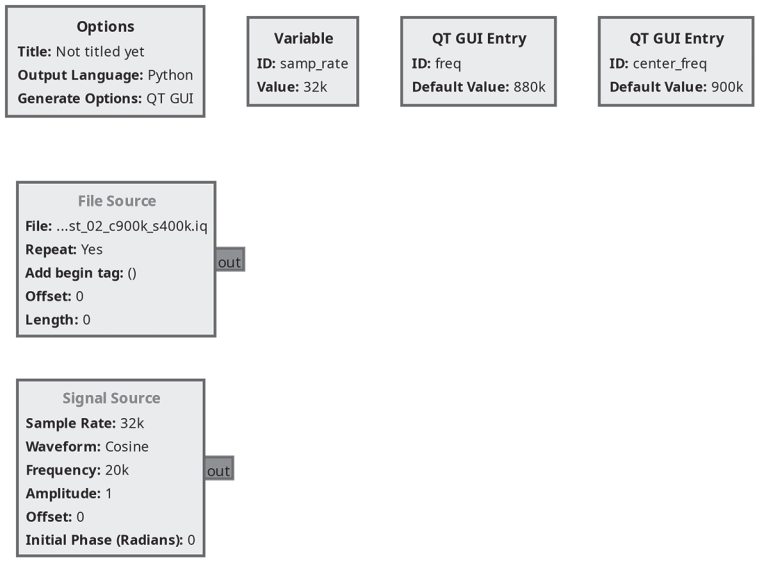


Figure 4-8: Adding the `Signal Source`

In the workspace, the `Signal Source` frequency is listed as `20k`, or 20,000. This is equal to the `center_freq` default value of `900e3` (900,000) minus the `freq` default value of `880e3` (880,000).

Next, place a `Multiply` block in the workspace and connect its inputs to your two sources (make sure you don't grab the wrong block, since there are a lot with the word *Multiply* in them). Remember how to connect blocks? Simply click the output port of the first block (in this case, one of the sources) and then click an input port of the second (in this case, the `Multiply` block). Clicking in reverse order also works. But which ports are the `Multiply` inputs? If you look closely at the text inside the tabs, you'll see that two of them say `in0` and `in1`. That's them! In general, the inputs will be on the left and the outputs on the right, but sometimes blocks will be rotated and this will no longer be true. When you're done, the flowgraph should look like Figure 4-9.

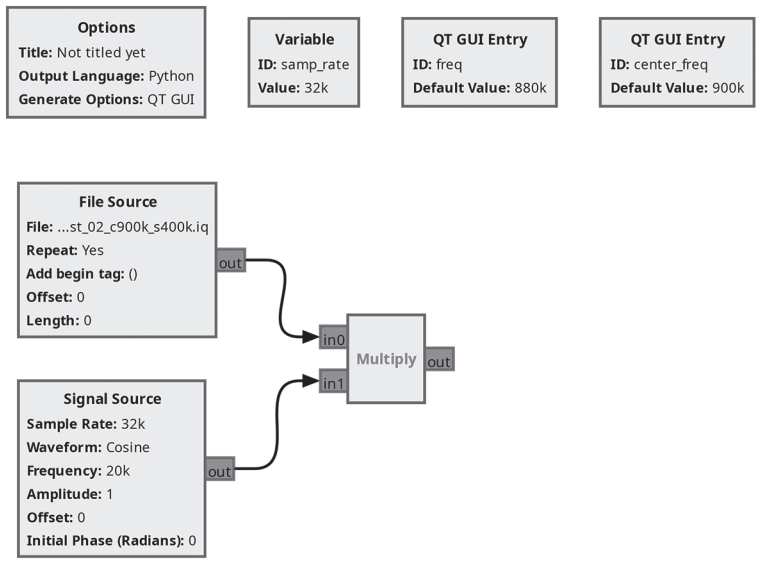


Figure 4-9: Adding and connecting the `Multiply` block

As you add and connect more blocks, it's good to take a moment here and there to tweak the positioning to tidy up your flowgraph. Just click and drag the blocks where you want them to go. As you do so, notice that the connections you've made are sticky and will follow your blocks wherever you drag them. This neatening process isn't required but can make your flowgraph much easier to read.

Next, add a `Low Pass Filter` block, setting the Cutoff Freq to `5e3` and the Transition Width to `1e3`, as shown in Figure 4-10. Leave all the rest of the properties alone.

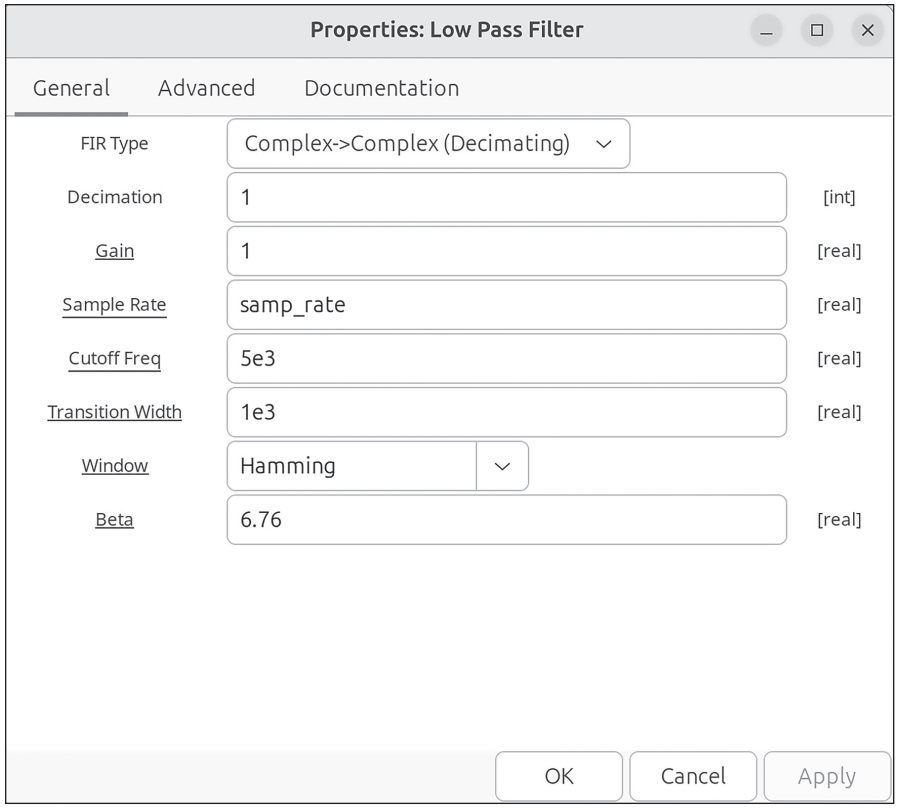


Figure 4-10: The `Low Pass Filter` properties

Filters like this block remove or reduce certain frequencies from a signal. They're an extremely important concept that we'll explore in detail in Chapter 5. Connect the `Low Pass Filter` input to the `Multiply` output. You should now have a workspace similar to Figure 4-11.

So far, here's what's happening in the flowgraph: the `File Source` block brings in data, which is then processed using the other three blocks (the `Signal Source`, `Multiply`, and `Low Pass Filter`). As we'll discuss in Chapter 6, these three blocks comprise the AM radio's tuner. They're what will allow you to focus in on individual radio channels within the source data.

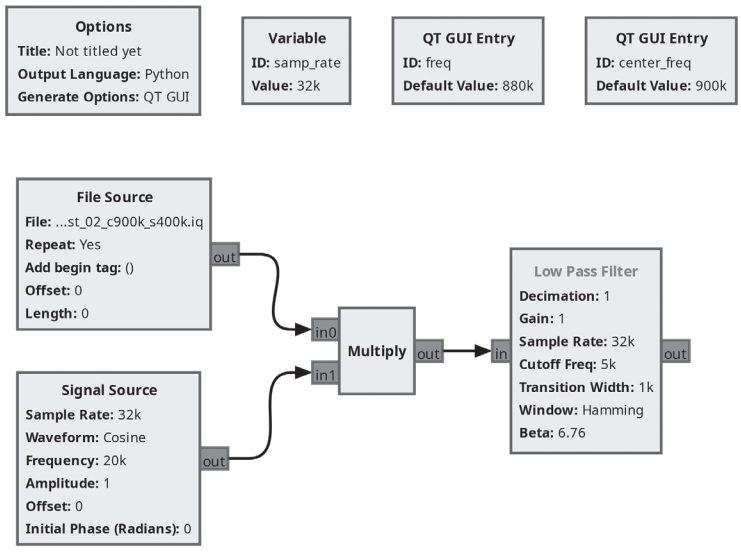


Figure 4-11: Adding the `Low Pass Filter` block

At this stage, the `Low Pass Filter` block's label should be red ~~(not shown in Figure 4-11, as the book is grayscale)~~, while the other block labels are black. This happens because GNU Radio Companion actively checks your flowgraph for errors as you build it, and any blocks with illegal conditions have their title displayed in red text. The illegal condition in this case is that the `Low Pass Filter` block's output isn't hooked up to anything. Other common illegal conditions include missing or invalid properties or multiple outputs connected to the same input.

Now place an `AM Demod` block and set its Channel Rate to `samp_rate` and its Audio Decimation to `1`, as shown in Figure 4-12.

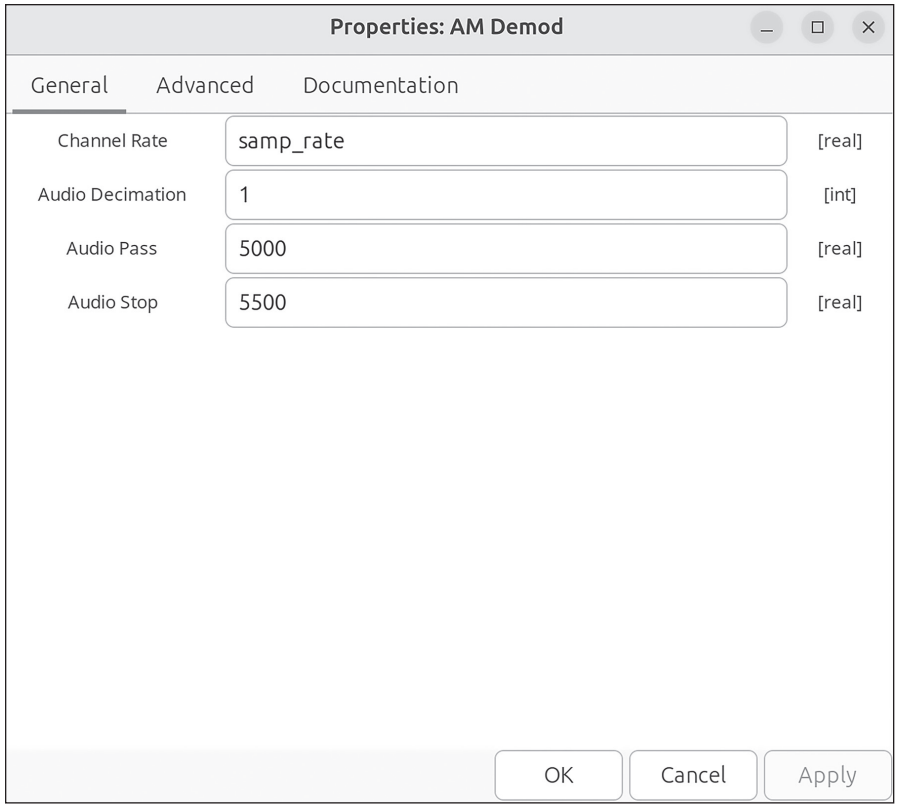


Figure 4-12: The `AM Demod` block properties

The `AM Demod` block will demodulate the incoming radio signal, extracting a human-understandable audio signal from it. We'll explore how its settings work in Chapter 6. Connect its input to the `Low Pass Filter` output, as shown in Figure 4-13.

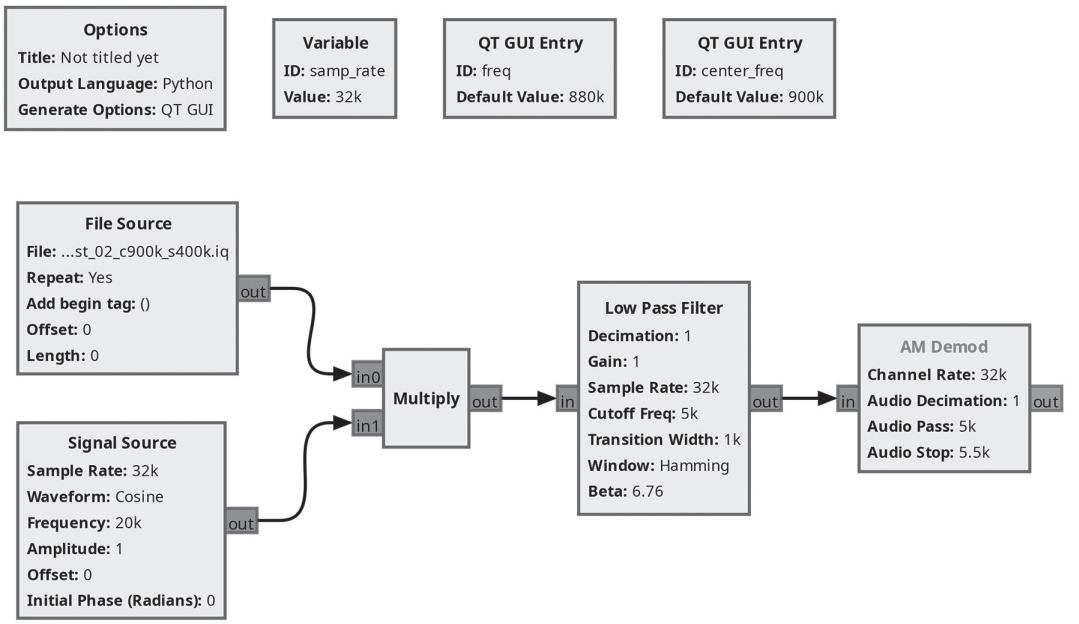


Figure 4-13: Adding the `AM Demod` block

If you're following along on your computer, notice that the input port of the `AM Demod` block is blue, while its output port is orange. This is significant: the colors represent the type of data that flows into or out of the block. Orange ports mean floating-point numbers are flowing, while blue ports mean complex numbers. This is definitely *not* the right time to get into complex numbers, so for now, just be aware that connections can only be made between ports of the same color.

Next, place a `Rational Resampler` block and set its Interpolation to `32` and the Decimation to `400`, as shown in Figure 4-14.

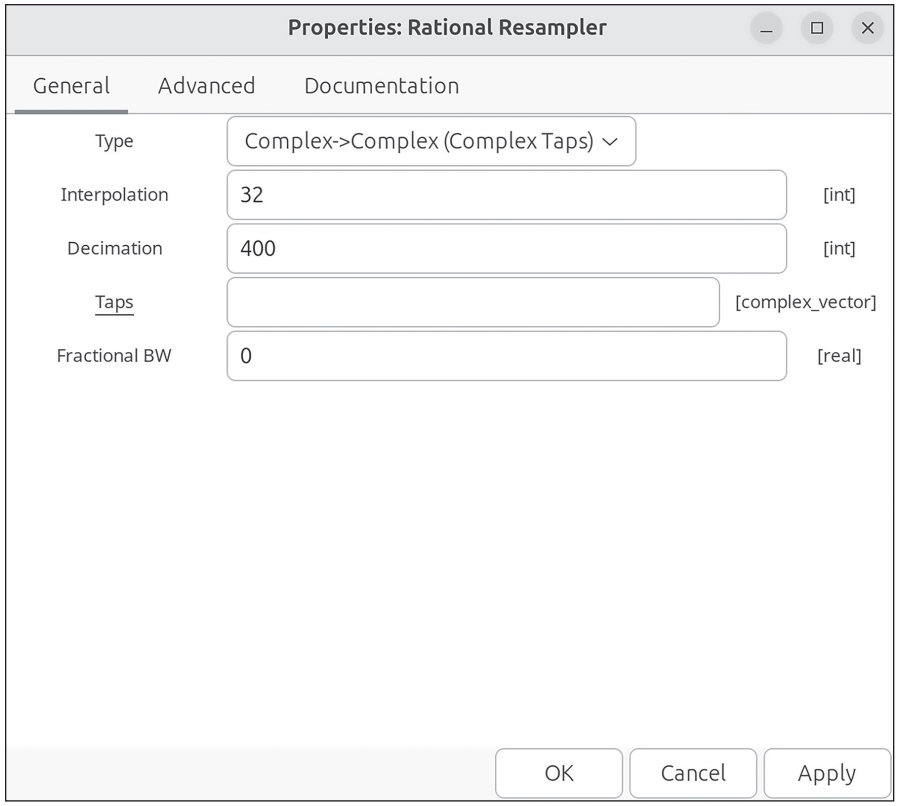


Figure 4-14: The `Rational Resampler` properties

The `Rational Resampler` block will adjust the sample rate of the audio signal so that your sound card can play it. Connect its input to the `AM Demod` output, like you see in Figure 4-15.

But not so fast! The connection should appear on your screen as a red arrow. Something is wrong, and there was a hint as to what a couple of paragraphs ago.

You can't connect ports of different colors. Thinking again in terms of programming languages, this would be like passing a string parameter to a function that's expecting an integer. To fix the problem, open up the `Rational Resampler` properties again and note the Type property is currently `Complex->Complex (Complex Taps)`. Instead, select **`Float->Float (Real Taps)`**, then click **OK**. You should now see that the `Rational Resampler` block's ports are orange, and the connection from the `AM Demod` to the `Rational Resampler` should have changed from red to black.

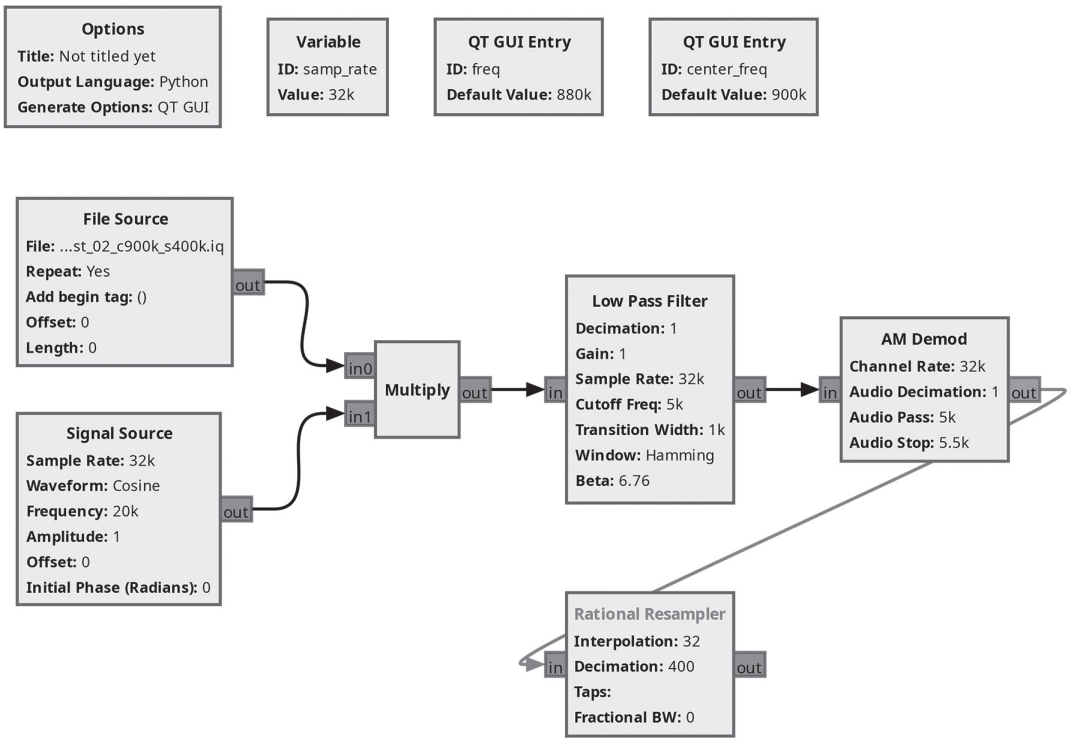


Figure 4-15: Adding the `Rational Resampler` block

We're almost done with the flowgraph. Take a moment to think about what might be missing. You have a source (two of them actually), and you have some blocks that process the data coming in from that source. But what are you going to do with your processed data stream? You need to dump it into a sink!

The Output

To output the data from the flowgraph, select an **`Audio Sink`** and add it to your workspace. This block will "play" the data through your computer's sound card so that you can hear the sounds being broadcast. Double-click the block and select a value of **`32kHz`** from the pull-down menu for the Sample Rate property. Then connect the output of the `Rational Resampler` to the input of the `Audio Sink`. With a bit of flowgraph tweaking, you could select a different sample rate, but most computer audio hardware supports the 32 kHz rate.

There's one last thing to do: change the sample rate for the flowgraph. Did you notice how most of the blocks have a Sample Rate of `32k` displayed? This is because all the blocks had a default Sample Rate property equal to

`samp_rate`. If you remember, a `Variable` block was present when you started the project. It had an ID of `samp_rate` and a value of `32000`. Double-click this variable and change its value to `400e3`, then watch what happens. You can also see the result in Figure 4-16.

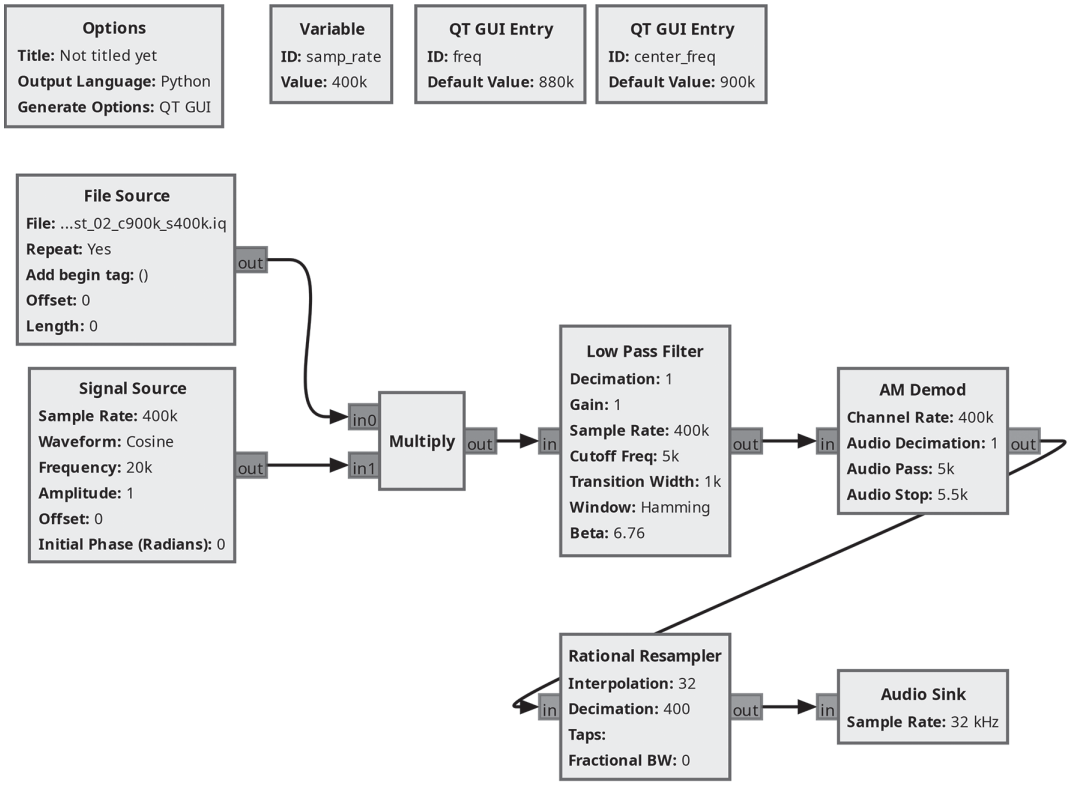


Figure 4-16: The completed flowgraph

Changing a single `Variable` block caused changes to ripple throughout the design such that all the blocks now have a sample rate of `400k`. All but the last one, anyway.

Run the flowgraph to see if it works. As in the last chapter, just click the toolbar icon that looks like a play button. If you hover over the button, it will say "Execute the flowgraph."

When you execute, some text will start scrolling by in the console window pane. Then, after a few seconds, you should hear some music. There's a bit of static, as you may have heard on other AM radios, but there are clearly identifiable voices. The music will loop after a few seconds because there's not a lot of data in the `File Source`, and it's set to repeat. If you don't hear any audio, try adjusting the speakers on your computer.

If you experience choppy audio and are running GNU Radio Companion in a virtual machine, open the properties window of the `Audio Sink` and change the Device Name to `sysdefault`, as you can see in Figure 4-17. If you aren't working in a VM, this isn't advised.

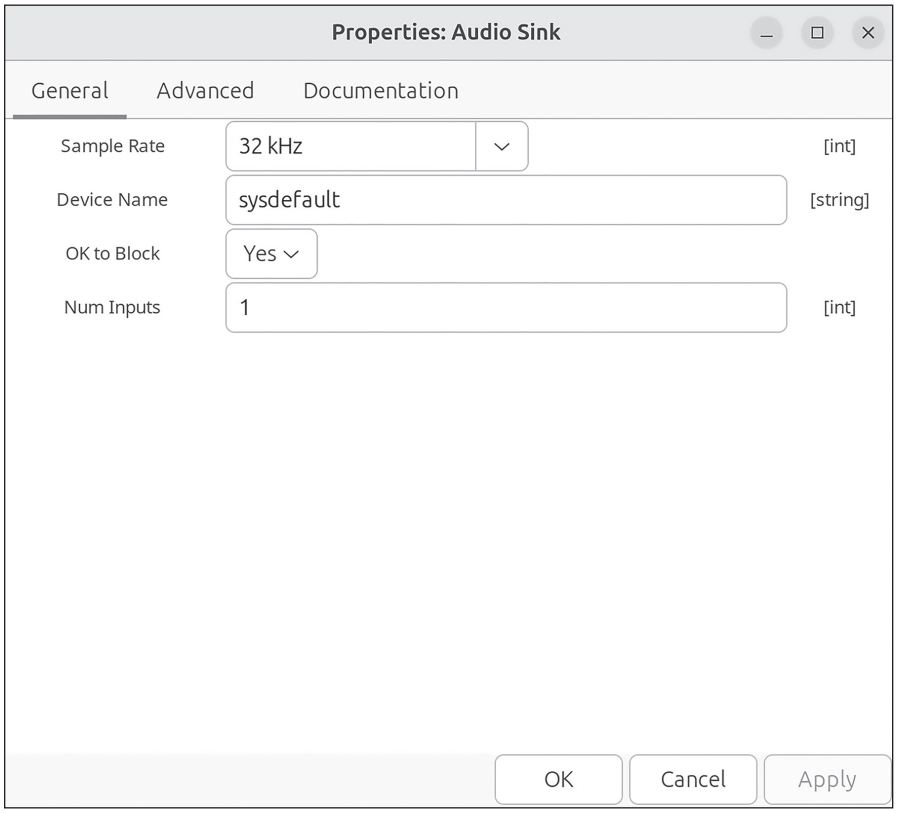


Figure 4-17: The `Audio Sink` setup for virtual machines

As the flowgraph runs, a window will pop up with the two `QT GUI Entry` elements in it, as shown in Figure 4-18. This is the *execution window*.

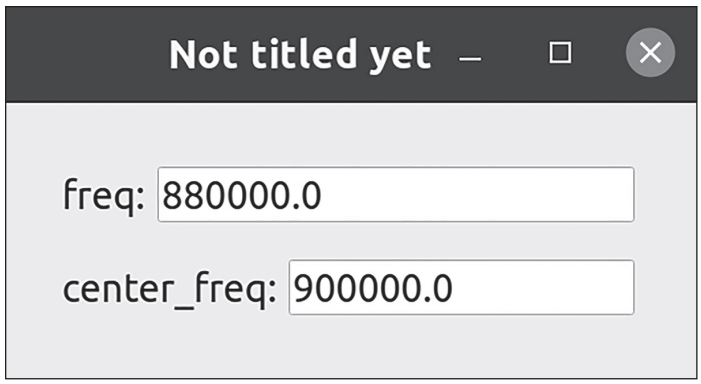


Figure 4-18: The flowgraph execution window

As mentioned early in this project, you can change these `QT GUI Entry` values while the radio is running, whereas to change an ordinary `Variable` block, like `samp_rate`, you'll have to stop the flowgraph, change the value, and run it again. In the execution window, change the value of `freq` to `750k`, then press ENTER. After a moment, you'll hear a different repeating chunk of audio. That's because you've now tuned to a different radio station! You can also find stations at `710k`, `750k`, and `1000k`, as well as a few fainter ones at `950k` and `1090k`.

Take a moment to think about what you just experienced. The person who created the input file you used didn't just record the *audio* from an AM station. That's pretty easy to do. They recorded the raw *radio signal* from that station, from which your flowgraph extracted recognizable audio. What's more, they didn't just record the radio signal broadcast on a *single* AM channel. They recorded the signals on a whole bunch of channels so that you can tune to any of them whenever you want.

This input file (and your flowgraph) provides your first glimpse of an extraordinary SDR capability: the ability to capture raw radio data and process it at will.

Conclusion

In this chapter, you built a simple AM radio receiver in GNU Radio Companion. Without going into a lot of detail yet, let's peel back one layer of the onion on your radio design. At a very high level, here's what your AM radio receiver flowgraph is doing:

1. Injecting prerecorded radio data into the flowgraph
2. Tuning to a specific AM radio channel while filtering out other AM radio channels
3. Demodulating the signal of your desired channel
4. Doing some magical thing called *resampling*
5. Playing the resulting audio on your computer speakers

The blocks responsible for each of these five tasks are highlighted in Figure 4-19.

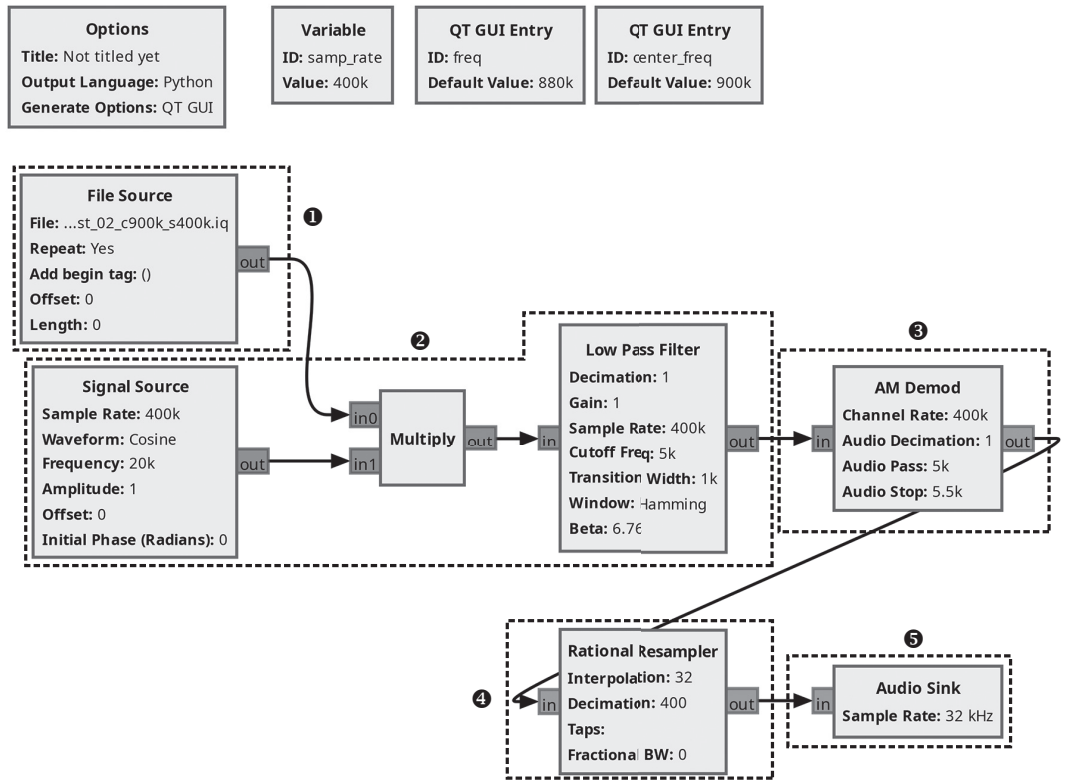


Figure 4-19: Breaking down the flowgraph by task

The `File Source` brings the data into the flowgraph ❶. The `Signal Source`, `Multiply`, and `Low Pass Filter` blocks work together to tune the signal ❷, which is then demodulated as it passes through the `AM Demod` block ❸. Then the signal is resampled by the `Rational Resampler` ❹ before being output by the `Audio Sink` ❺.

You probably have a few questions. How does a filter work? What does it even mean to resample? How on earth does multiplying a radio signal by a sinusoid tune anything? You'll get the answers these questions in the next few chapters, partly so you can understand how your AM radio flowgraph works, but even more so because the answers will illuminate some critical radio concepts.