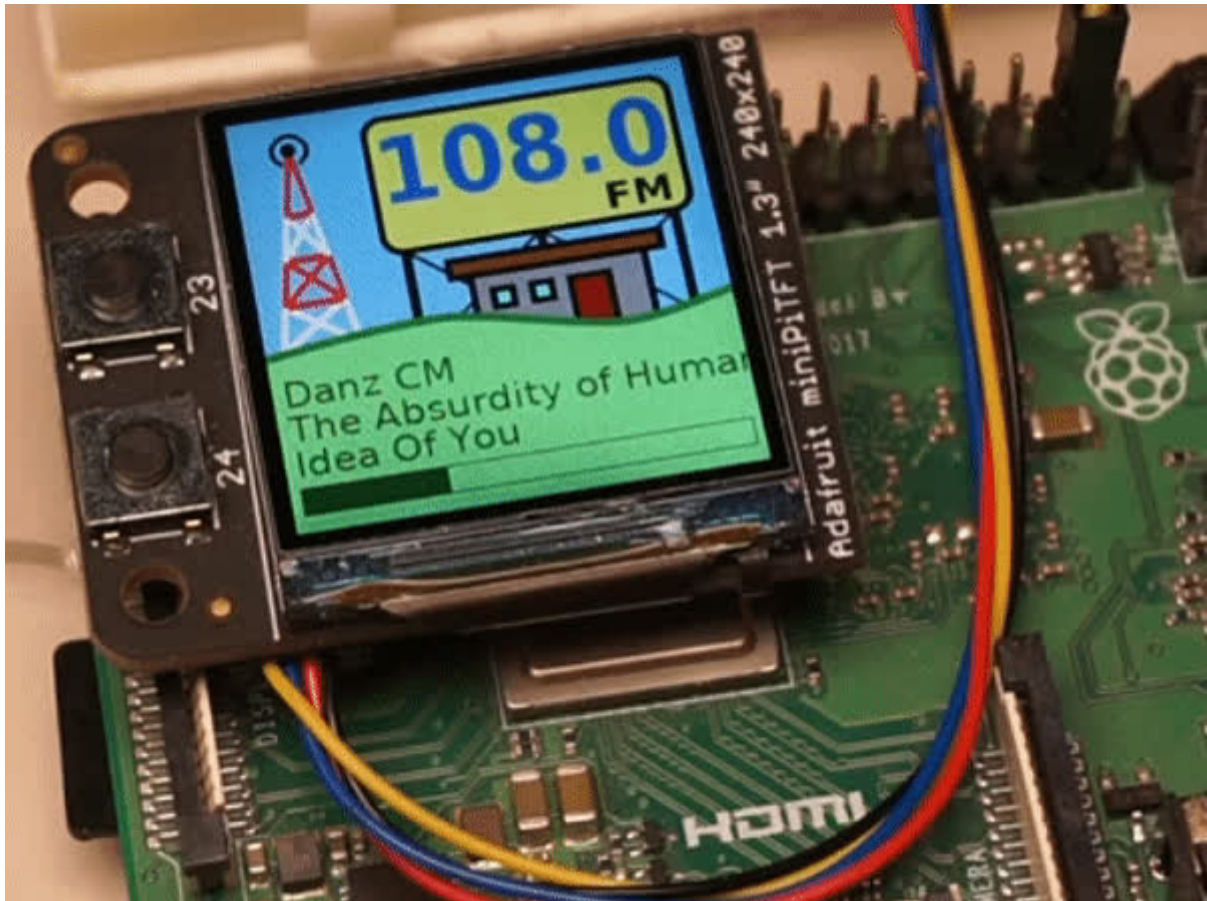




PiPyPirate Radio

Created by Carter Nelson



<https://learn.adafruit.com/pipypirate-radio>

Last updated on 2024-03-08 04:13:56 PM EST

Table of Contents

Overview	3
<ul style="list-style-type: none">• FM Radio? Really?• Parts	
Hardware Setup	5
Software Setup	8
<ul style="list-style-type: none">• Blinka Installation• PiTFT Setup• FM Radio Setup• USB Audio Adapter Setup• Music Player Software• Python Library for MPD	
MPD Configuration	12
<ul style="list-style-type: none">• Music and Playlists Folders• MPD Configuration• Restart MPD Server	
Adding Music	15
<ul style="list-style-type: none">• Add Music Files• Create Playlist• Update MPD	
What's The Frequency, Kenneth?	18
Radio Code	19
<ul style="list-style-type: none">• Background Image• Configure and Run• Info and Status• Playback Control	

Overview



So called "[pirate" radio stations](https://adafru.it/18Ek) (<https://adafru.it/18Ek>) have existed as long as radio broadcast has existed. These unlicensed operators setup shop in various locations and start broadcasting whatever content they wished. Shown above is the lightship used as a base for [Radio Veronica](https://adafru.it/18EI) (<https://adafru.it/18EI>) broadcasting offshore of the Netherlands in the 1960's. Being based offshore on a boat makes it extra pirate-y!

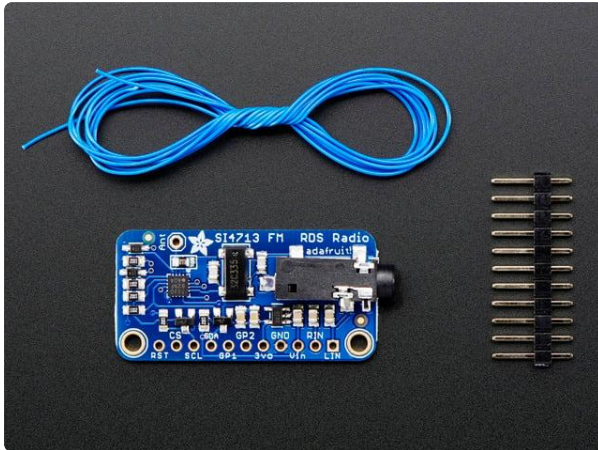
In this guide we'll show how to use the [Adafruit Si4713 FM Transmitter](http://adafru.it/1958) (<http://adafru.it/1958>) to create your own little pirate radio station. A Raspberry Pi provides the source for music storage and playback. Attaching a small [1.3" PiTFT](http://adafru.it/4484) (<http://adafru.it/4484>) provides a display for playback song information and status.

FM Radio? Really?

Pretty much no one carries a personal FM radio receiver with them these days. So if you're all like pepperidge-farm-remembers / i-was-there-gandalf at the mention of FM radio and wondering what's the point, here are some ideas:

- Create a personal radio station for someone that does not have a smartphone. Perhaps for an older generation member more comfortable with FM radio technology, [as was heartwarmingly done in this post](https://adafru.it/18Em) (<https://adafru.it/18Em>).
- Broadcast audio to cars at an ad hoc "drive in" style gathering. Cars still have FM radio receivers.
- Other imaginative use :)

Parts



Adafruit Stereo FM Transmitter with RDS/RBDS Breakout - Si4713

Yaaar! Become your very own pirate radio station with this FM radio transmitter. This breakout board, based on the best-of-class Si4713, is an all-in-one stereo audio FM transmitter...

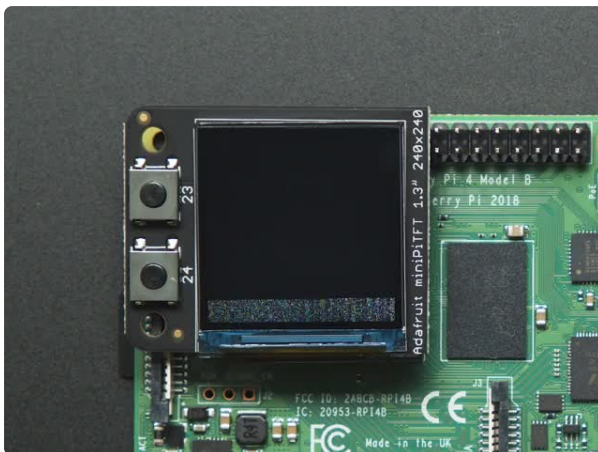
<https://www.adafruit.com/product/1958>



USB Audio Adapter - Works with Raspberry Pi

The Raspberry Pi has an on-board audio jack, which is super handy for all kinds of sound effects and speech, just plug and go! However, for when you want better audio for music...

<https://www.adafruit.com/product/1475>



Adafruit Mini PiTFT 1.3" - 240x240 TFT Add-on for Raspberry Pi

If you're looking for the most compact li'l color display for a Raspberry Pi (most likely a

<https://www.adafruit.com/product/4484>



Raspberry Pi 3 - Model B - ARMv8 with 1G RAM

Did you really think the Raspberry Pi would stop getting better? At this point, we sound like a broken record, extolling on the new Pi's myriad improvements like we're...

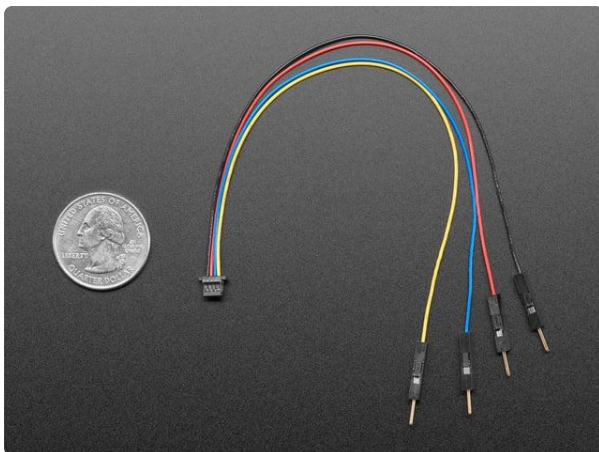
<https://www.adafruit.com/product/3055>



Stereo 3.5mm Plug/Plug Audio Cable - 6 feet

This basic cable comes with two 3.5mm (1/8" headphone jack size) stereo connectors. It's fairly straight forward, you'll commonly need these to connect two audio devices...

<https://www.adafruit.com/product/876>



STEMMA QT / Qwiic JST SH 4-pin to Premium Male Headers Cable

This 4-wire cable is a little over 150mm / 6" long and fitted with JST-SH female 4-pin connectors on one end and premium Dupont male headers on the other. Compared with the...

<https://www.adafruit.com/product/4209>

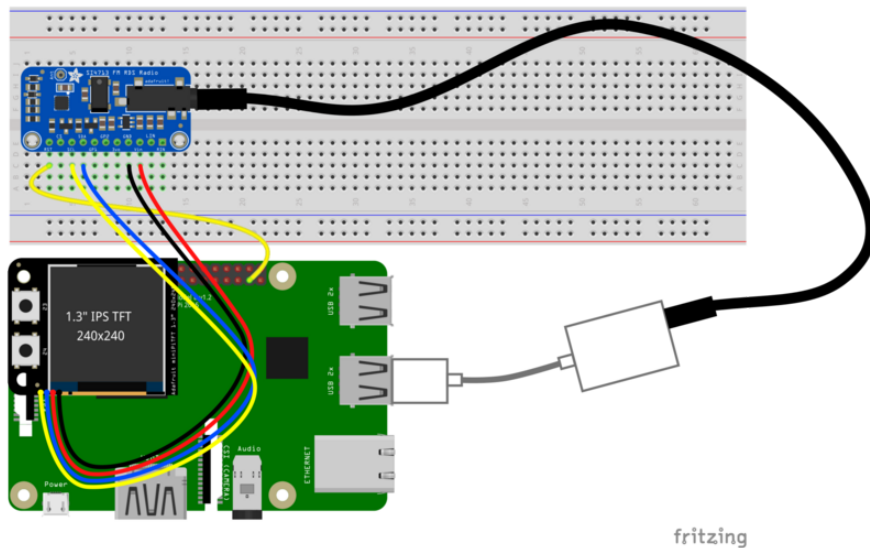
Hardware Setup

In this guide we demonstrate using a Raspberry Pi Model 3B. However, the processing being done is minimal, and actually any model Pi could probably handle this task. A USB port makes connecting the USB audio adapter easy.

The 1.3" PiTFT used is another trade-off. It's pretty small and only provides two buttons for user interaction. But it leaves some of the Pi's GPIO pins available, which

are needed for connecting the Si4713's reset line. Also, it has a STEMMA QT connector which makes the I2C connections easy.

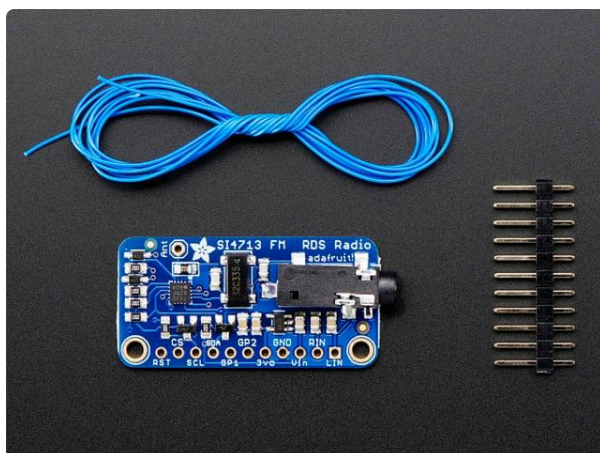
Here's a connection diagram of the overall hardware setup.



- Attach the **PiTFT** to the **Pi GPIO** header.
- Use the **STEMMA QT** connector on the PiTFT to connect to power and I2C to the Si4713.
- Connect **Si4713 RST** to **GPIO 26** on the Pi.
- Plug the **USB audio adapter** into an available USB port on the Pi.
- Connect the USB audio adapter to the Si4713 input using an **audio cable**.

Here is a list of the various hardware items shown.

The Si4714 is the FM radio transmitter:



[Adafruit Stereo FM Transmitter with RDS/ RBDS Breakout - Si4713](https://www.adafruit.com/product/1958)

Yaaar! Become your very own pirate radio station with this FM radio transmitter. This breakout board, based on the best-of-class Si4713, is an all-in-one stereo audio FM transmitter...

<https://www.adafruit.com/product/1958>

For getting good audio out from the Raspberry Pi, a USB audio adapter is used:



USB Audio Adapter - Works with Raspberry Pi

The Raspberry Pi has an on-board audio jack, which is super handy for all kinds of sound effects and speech, just plug and go! However, for when you want better audio for music...

<https://www.adafruit.com/product/1475>

To connect the USB audio adapter to the FM radio, a 3.5mm stereo plug/plug cable is needed. This cable is nothing special and one is often included with various media devices. So check your electronic drawer first - you may already have one.

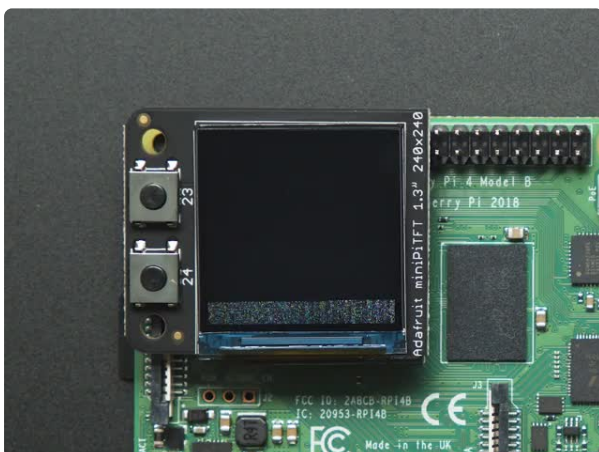


Stereo 3.5mm Plug/Plug Audio Cable - 6 feet

This basic cable comes with two 3.5mm (1/8" headphone jack size) stereo connectors. It's fairly straight forward, you'll commonly need these to connect two audio devices...

<https://www.adafruit.com/product/876>

The 1.3" PiTFT provides status and buttons for user interaction:

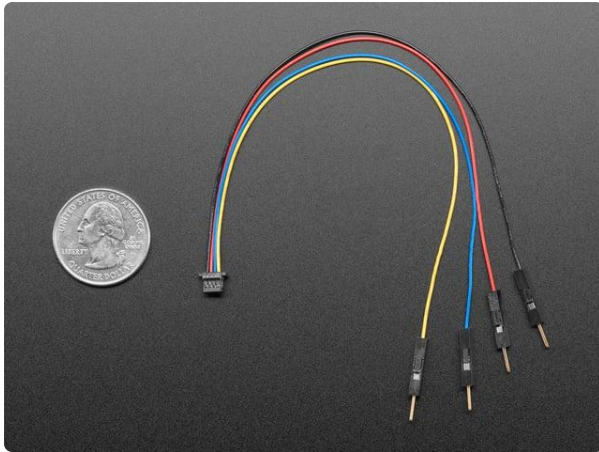


Adafruit Mini PiTFT 1.3" - 240x240 TFT Add-on for Raspberry Pi

If you're looking for the most compact li'l color display for a Raspberry Pi (most likely a

<https://www.adafruit.com/product/4484>

The Si4713 is connected to the Pi through the PiTFT's STEMMA QT connector. This cable can be used:



STEMMA QT / Qwiic JST SH 4-pin to Premium Male Headers Cable

This 4-wire cable is a little over 150mm / 6" long and fitted with JST-SH female 4-pin connectors on one end and premium Dupont male headers on the other.

Compared with the...

<https://www.adafruit.com/product/4209>

And of course, need a Raspberry Pi. We based this guide on the Pi 3 Model B:



Raspberry Pi 3 - Model B - ARMv8 with 1G RAM

Did you really think the Raspberry Pi would stop getting better? At this point, we sound like a broken record, extolling on the new Pi's myriad improvements like we're...

<https://www.adafruit.com/product/3055>

The setup above also shows a breadboard. But direct connections to the Si4713 is also possible.

Software Setup

OK, let's get all the necessary software bits installed and checked out. The first few are covered in other guides, which are linked to from here. It works best to do these in the order shown here, checking that each step works before moving to the next.

Use the Lite version of the Raspberry Pi OS.

Blinka Installation

Follow this guide page for initial Pi setup (including the OS) and Blinka installation:

Blinka Pi Installation

<https://adafru.it/Deo>

Be sure the **blinkatest.py** script from that setup runs as expected before moving on.

PiTFT Setup

This guide uses the PiTFT directly via Python. So follow this setup page from the PiTFT main guide:

PiTFT Setup

<https://adafru.it/HBL>

The guide covers a couple of TFTs, and the **rgb_display_minipitfttest.py** test script appears to be configured for the smaller TFT - **not the one used in this guide**.

However, it's a simple fix to change to the 1.3" TFT.

Change these lines:

```
display = st7789.ST7789(  
    board.SPI(),  
    cs=cs_pin,  
    dc=dc_pin,  
    rst=reset_pin,  
    baudrate=BAUDRATE,  
    width=135,  
    height=240,  
    x_offset=53,  
    y_offset=40,  
)
```

to this:

```
disp = st7789.ST7789(  
    board.SPI(),  
    cs=cs_pin,  
    dc=dc_pin,  
    rst=reset_pin,  
    baudrate=BAUDRATE,  
    width=240,  
    height=240,  
    x_offset=0,  
    y_offset=80,  
)
```

Don't skip the DejaVu TFT Font installation done in this guide. Those fonts are used again later.

FM Radio Setup

To install the CircuitPython library for the Si4713 FM radio transmitter, follow this guide page:

Si4713 Setup

<https://adafru.it/18En>

Test using the frequency scan example from that guide. If that runs OK, then it should be installed and working correctly.

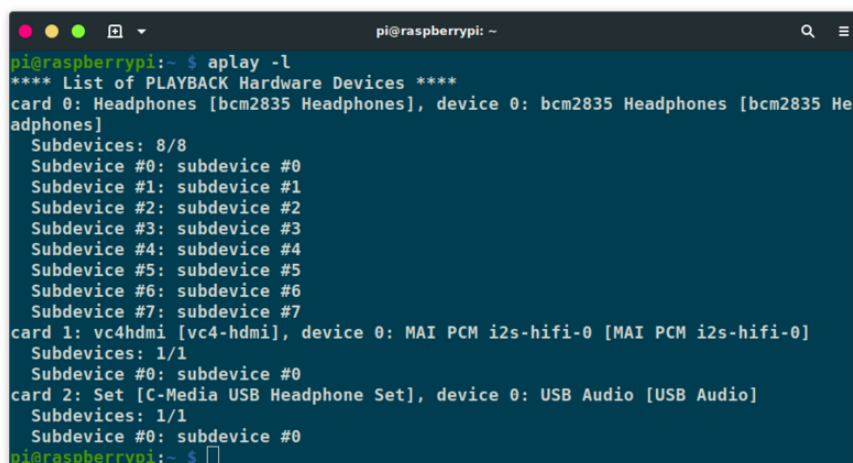
USB Audio Adapter Setup

The Pi has built in support for the [USB audio adapter](http://adafru.it/1475) (<http://adafru.it/1475>), so there's nothing extra needed in terms of software. Simply plug the USB audio adapter into one of the Pi's USB ports and run the following command:

```
aplay -l
```

This will list the audio devices that the Pi has. There will likely be more than one, but the USB audio adapter should show up in the list. Look for **C-Media USB Headphone Set** in the output.

Here's an example:



```
pi@raspberrypi: ~  
pi@raspberrypi:~$ aplay -l  
**** List of PLAYBACK Hardware Devices ****  
card 0: Headphones [bcm2835 Headphones], device 0: bcm2835 Headphones [bcm2835 Headphones]  
  Subdevices: 8/8  
  Subdevice #0: subdevice #0  
  Subdevice #1: subdevice #1  
  Subdevice #2: subdevice #2  
  Subdevice #3: subdevice #3  
  Subdevice #4: subdevice #4  
  Subdevice #5: subdevice #5  
  Subdevice #6: subdevice #6  
  Subdevice #7: subdevice #7  
card 1: vc4hdmi [vc4-hdmi], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]  
  Subdevices: 1/1  
  Subdevice #0: subdevice #0  
card 2: Set [C-Media USB Headphone Set], device 0: USB Audio [USB Audio]  
  Subdevices: 1/1  
  Subdevice #0: subdevice #0  
pi@raspberrypi:~$
```

The USB audio adapter shows up as **card 2**.

Take note of what card number the USB audio adapter shows up as. This may be needed later.

Music Player Software

To serve up audio, we'll use the [Music Player Daemon \(https://adafru.it/18Eo\)](https://adafru.it/18Eo) software. To install it, along with a couple of other tools, run the following:

```
sudo apt-get install mpd mpc ncmtc
```

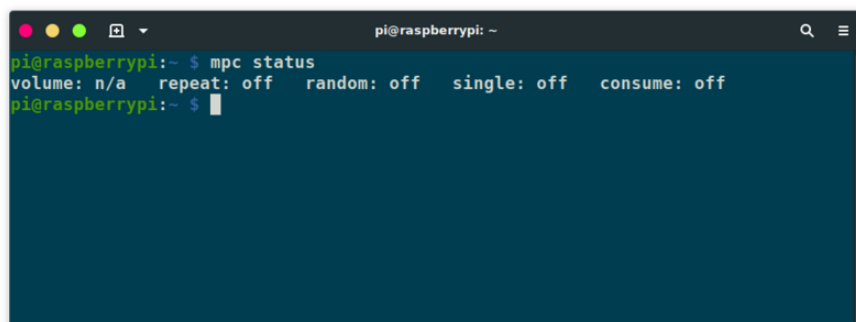
This will install:

- **mpd** - the main music server (daemon)
- **mpc** - a simple command line tool for controlling mpd
- **ncmtc** - a simple text based interface music player

After installation, run the following as a simple test:

```
mpc status
```

This just queries the current status of the server, which won't be anything exciting at this point. It should look like this:



But that verifies the server has been installed correctly and is running.

Python Library for MPD

The [python-mpd2 \(https://adafru.it/18Ep\)](https://adafru.it/18Ep) library allows for controlling mpd playback from within Python. This is another pip install:

```
sudo pip3 install python-mpd2
```

MPD Configuration

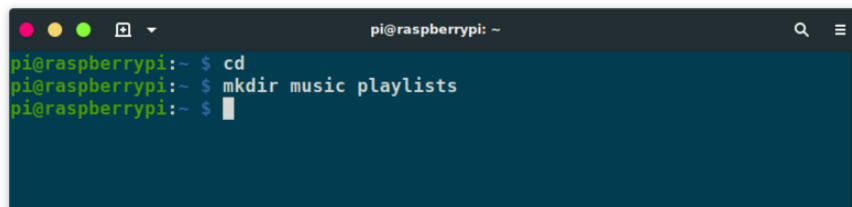
Music and Playlists Folders

We'll store music and playlist files in the default **pi** users home directory. Use **mkdir** to create these folders. An initial **cd** is done to make sure the current directory is the **pi** user's home directory (**/home/pi**).

Run these commands:

```
cd
mkdir music playlists
```

Nothing exciting should happen.



- The **/home/pi/music** directory will contain the actual music files, like MP3s, etc.
- The **/home/pi/playlists** directory will contain playlists.

MPD Configuration

The main file that controls the mpd configuration is located in **/etc/mpd.conf**. A default file is added during the installation of mpd. There is a lot of content in this file, however the vast majority is commented out and is just there for reference. For this guide, it is possible to use a very minimal configuration.

First, let's move (rename) the original file so it's still available as a backup:

```
sudo mv /etc/mpd.conf /etc/mpd_orig.conf
```

Now use a text editor to add the contents below to a new empty **/etc/mpd.conf**.

```
music_directory      "/home/pi/music"
playlist_directory    "/home/pi/playlists"

audio_output {
    type              "alsa"
    name              "USB Audio Adapter"
```



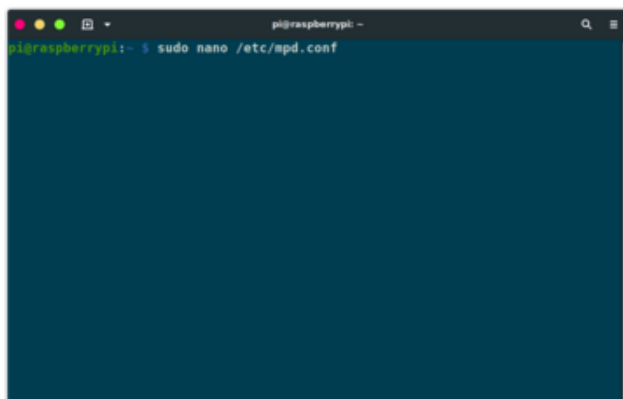
```
}      device      "hw:2,0"
```

This sets the following:

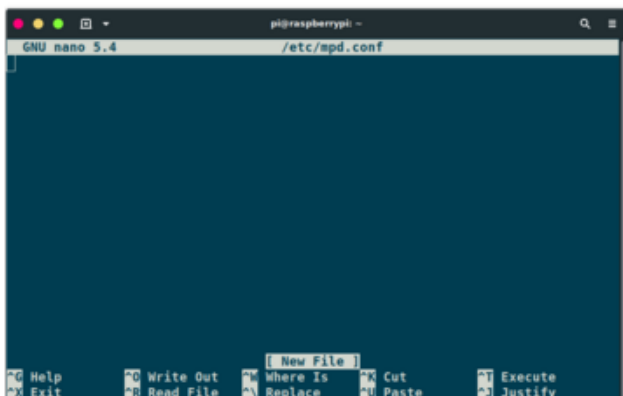
- Sets the **music_directory** and **playlist_directory** to the locations we created previously.
- The **audio_output** lines enable using the USB audio adapter. Actually selecting this output is done later.

The 2 in "hw:2,0" is the card number for the USB audio adapter. Change this number as needed based on `aplay -l` output.

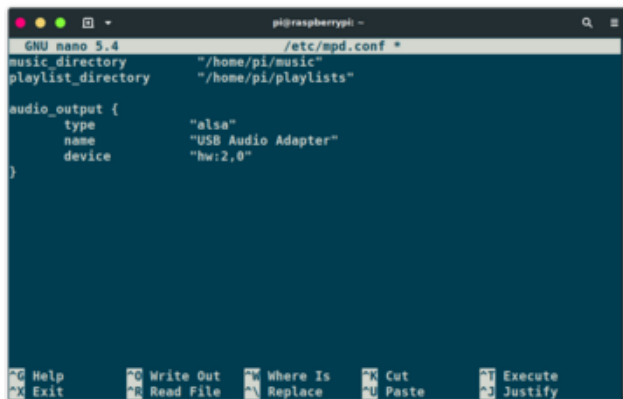
Adding this content to `/etc/mpd.conf` can be done using the **nano** text editor as shown below.



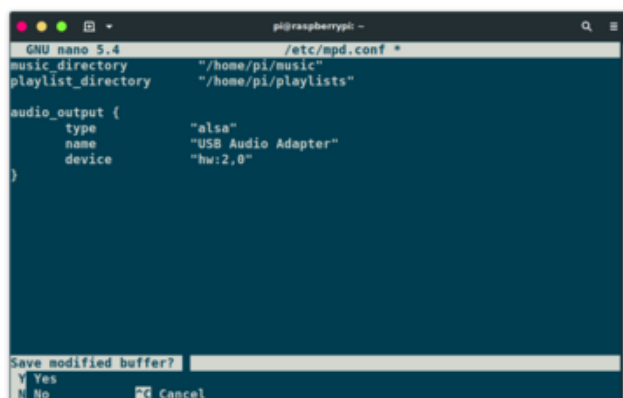
Use the command `sudo nano /etc/mpd.conf` to open the nano editor on a new file.



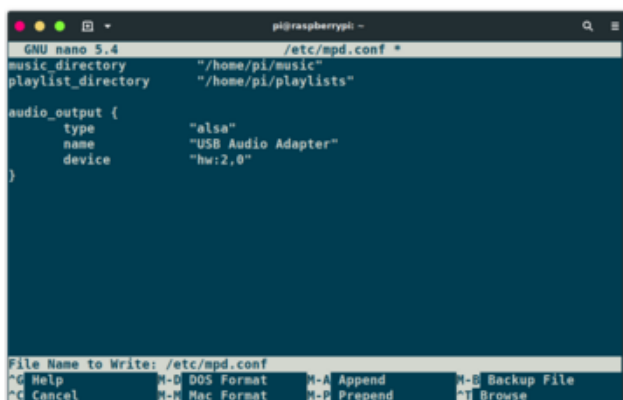
The contents should initially be empty. If not, make sure a backup was made, and then just delete everything.



Now copy the configuration file contents provided above and paste it into the file.



Press <CTRL><X> to exit. It will prompt to save the file. Press Y and hit <ENTER>.



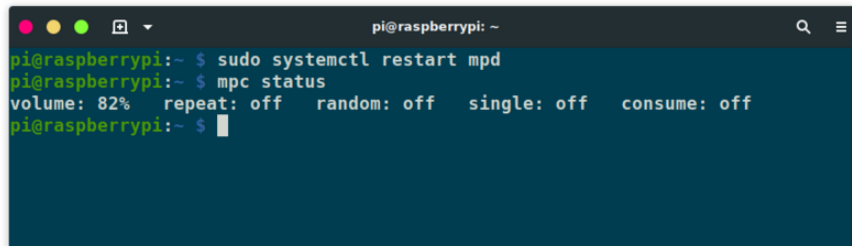
If nano asks for a file name, make sure it's /etc/mpd.conf and press <ENTER>

Restart MPD Server

For the configuration changes to take effect, the mpd server needs to be restarted. Use the following command:

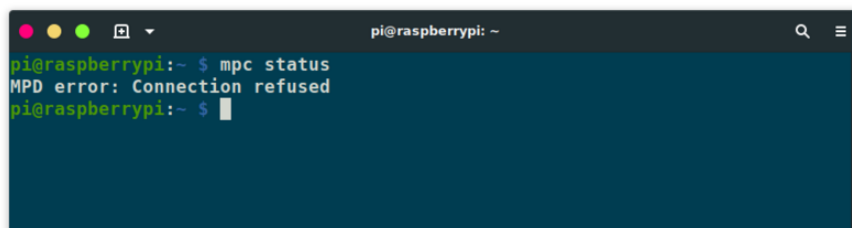
```
sudo systemctl restart mpd
```

This should **not** generate any additional output. So running `mpc status` as a quick sanity check can help to make sure the mpd server actually did restart.

A terminal window on a Raspberry Pi showing the command `sudo systemctl restart mpd` and `mpc status`. The output of `mpc status` is: `volume: 82% repeat: off random: off single: off consume: off`.

```
pi@raspberrypi: ~  
pi@raspberrypi:~$ sudo systemctl restart mpd  
pi@raspberrypi:~$ mpc status  
volume: 82% repeat: off random: off single: off consume: off  
pi@raspberrypi:~$
```

For comparison, here is what `mpc status` will output if the mpd server is **not** running:

A terminal window on a Raspberry Pi showing the command `mpc status`. The output is: `MPD error: Connection refused`.

```
pi@raspberrypi:~$ mpc status  
MPD error: Connection refused  
pi@raspberrypi:~$
```

Adding Music

There are two steps for adding music:

- Add the music files (mp3, etc.) to the `/home/pi/music` directory.
- Create playlists files (m3u) in the `/home/pi/playlists` directory.

Playlists are really an optional feature from mpd's point of view. However, the Python radio program works by specifying a playlist to use for broadcast. So we'll go through both steps.

Add Music Files

This is pretty simple, just dump all the mp3 files in to the **music** directory:

```
music  
├── song1.mp3  
└── song2.mp3
```

However, if there are a lot of music files, from numerous artists and albums, then this can get messy. The mpd server will search the music folder recursively. So a good way to organize things is into a hierarchy of artist/album/song. Something like this:

```
music
├── artist1
│   ├── album1
│   │   ├── song1.mp3
│   │   └── song2.mp3
│   └── album2
│       └── song1.mp3
└── artists2
```

To provide examples for this guide, we'll use these two short mp3 files. Click each button to download them.

beats.mp3

<https://adafru.it/18Eq>

happy.mp3

<https://adafru.it/18Er>

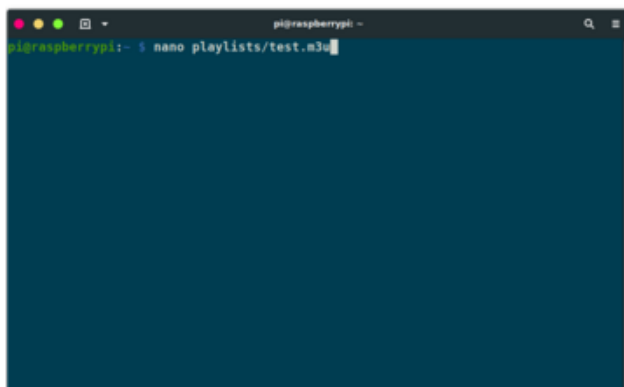
Copy them to the Raspberry Pi. To keep things simple, we'll just place them directly in the music directory so it ends up looking like this:

```
music
├── beats.mp3
└── happy.mp3
```

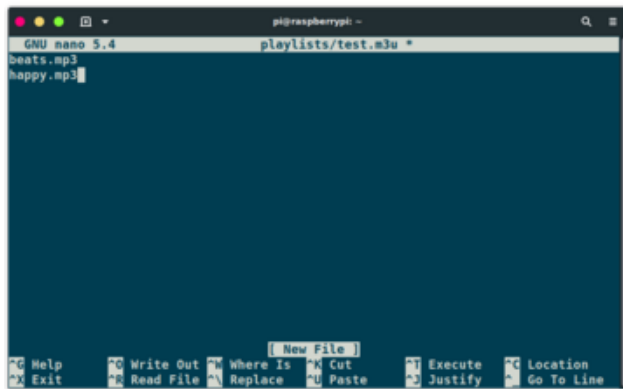
Create Playlist

A playlist is a simple text file with a .m3u extension. Each line of the playlist file references a music file. The full path of the file relative to the music folder is used.

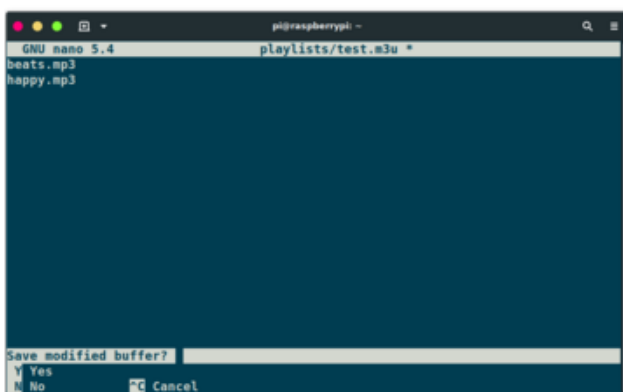
Let's create one called **test.m3u** using the two example files downloaded in the previous section.



Use nano to create the **test.m3u** playlist file in the **playlists** directory.



Add the names of the two mp3 files.



Press <CTRL><X> to save and exit.

Multiple playlist files can be created and stored in the **playlists** directory.

Update MPD

Anytime new music files are added, or playlists added or updated, the mpd server needs to be told to update its internal database. This can be done using mpc:

```
mpc update
```

This will generally be very fast. But if there are a lot of new files to be searched through, it can take many seconds. The update progress can be checked using the status command with mpc. If this does not show "Updating DB", then the update is complete.



What's The Frequency, Kenneth?

The Si4713 FM Radio Transmitter uses the same FM radio band as regular domestic radio. So a lot of frequencies will already be occupied by local radio stations - like all the ones you hear driving around in your car.

To find an available frequency, we can use the Si4713 itself, which has the built in ability to measure the noise for a given FM frequency. By scanning across the frequencies in the [FM broadcast band \(https://adafru.it/18Es\)](https://adafru.it/18Es), locations where frequencies are occupied can be determined.

Use the code below to run a frequency scan. This is really just the [same basic demo from the Si4713 guide \(https://adafru.it/18En\)](https://adafru.it/18En), with added output to a file.

```
# SPDX-FileCopyrightText: 2023 Carter N. for Adafruit Industries
# SPDX-License-Identifier: MIT

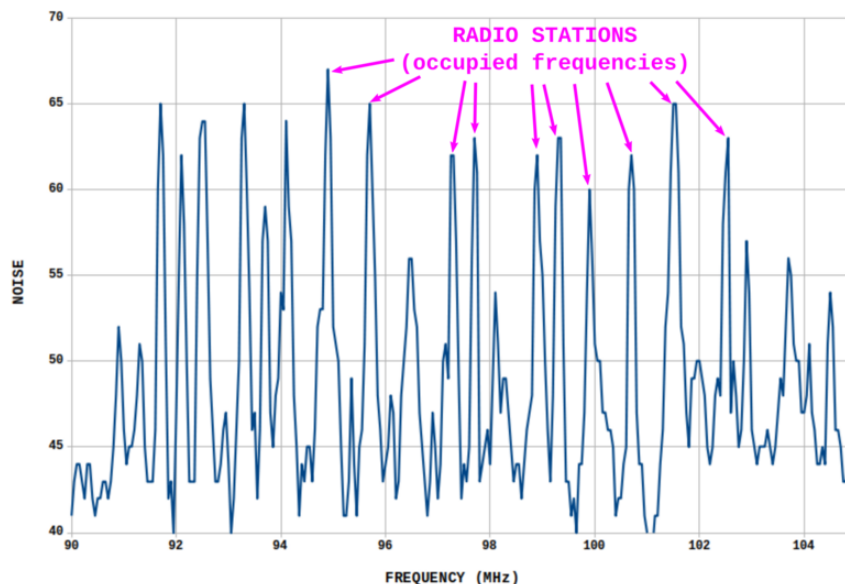
import board
import digitalio
import adafruit_si4713

radio = adafruit_si4713.SI4713(
    board.I2C(),
    reset=digitalio.DigitalInOut(board.D26),
    timeout_s=0.5
)

with open("freq_scan.dat", "w") as fp:
    for f_khz in range(87500, 108000, 50):
        noise = radio.received_noise_level(f_khz)
        fp.write("{}{}\n".format(f_khz/1000.0, noise))
        print('{0:0.3f} mhz = {1} dBuV'.format(f_khz/1000.0, noise))
```

The output is simple. For each frequency, a relative noise level is given. The higher the value, the stronger the radio station signal. So look for frequencies with the lowest values.

The data can also be plotted to help find the occupied areas. **Pick a frequency somewhere between the peaks.**



Radio Code

At this point, all the supporting software pieces should be in place, music has been added, playlist(s) created, and an available broadcast frequency has been determined. Now we can use a Python program to configure the Si4713 FM radio transmitter, start music playback, and use the TFT to provide status and basic control (via the buttons).

Here's the code. Save a copy of this as **radio.py** in the pi users home directory **/home/pi**.

The default code expects the music files and playlist added from the previous section to be in place.

```
# SPDX-FileCopyrightText: 2023 Carter N. for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
import digitalio
import adafruit_si4713
from PIL import Image, ImageDraw, ImageFont
from adafruit_rgb_display import st7789
import mpd

#--| User Config |-----
FREQ = 89.00
PLAYLIST = "test"
STATION_NAME = "PiPyPirate Radio"
UPDATE_RATE = 0.5
#-----

#==| SETUP |=====

# Display
disp = st7789.ST7789(
    board.SPI(),
```

```

        height=240,
        y_offset=80,
        rotation=180,
        cs=digitalio.DigitalInOut(board.CE0),
        dc=digitalio.DigitalInOut(board.D25),
        rst=digitalio.DigitalInOut(board.D24),
        baudrate=64000000,
    )

    backlight = digitalio.DigitalInOut(board.D22)
    backlight.switch_to_output()
    backlight.value = True

    background = Image.open("radio_bg.png")
    STAT_FNT = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSansCondensed-Bold.ttf", 55)
    STAT_CLR = (30, 100, 200)
    INFO_FNT = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 20)
    ARTS_CLR = (0, 100, 0)
    ALBM_CLR = (0, 100, 0)
    TITL_CLR = (0, 100, 0)
    PROG_CLR = (0, 100, 0)

    # Buttons
    button1 = digitalio.DigitalInOut(board.D23)
    button1.switch_to_input(pull=digitalio.Pull.UP)
    button2 = digitalio.DigitalInOut(board.D24)
    button2.switch_to_input(pull=digitalio.Pull.UP)

    # Radio
    radio = adafruit_si4713.SI4713(
        board.I2C(),
        reset=digitalio.DigitalInOut(board.D26),
        timeout_s = 0.5
    )
    radio.tx_frequency_khz = int(FREQ * 1000)
    radio.tx_power = 115
    radio.configure_rds(0xADAF, station=STATION_NAME.encode())

    # MPD
    mpc = mpd.MPDClient()
    mpc.connect("localhost", 6600)
    mpc.stop()
    mpc.clear()
    mpc.load(PLAYLIST)
    mpc.play()
    mpc.repeat(1)
    #=====

    def button1_handler():
        if status['state'] == 'play':
            mpc.pause()
        else:
            mpc.play()

    def button2_handler():
        mpc.next()

    def update_display():
        image = background.copy()
        draw = ImageDraw.Draw(image)

        draw.text(
            (150, 20),
            "{:>5.1f}".format(FREQ),
            anchor="mt",
            font=STAT_FNT,
            fill=STAT_CLR
        )

```



```

if status['state'] == 'play':
    r = 10 * (1 + int(time.monotonic() % 3))
    draw.arc( (30-r, 20-r, 30+r, 20+r),
              120, 60,
              fill = (0, 0, 0),
              width = 3
            )

info = mpc.currentsong()
artist = info.get('artist', 'unknown')
album = info.get('album', 'unknown')
song = info.get('title', 'unknown')
draw.text( (5, 150), artist, font=INFO_FNT, fill=ARTS_CLR )
draw.text( (5, 170), album, font=INFO_FNT, fill=ALBM_CLR)
draw.text( (5, 190), song, font=INFO_FNT, fill=TITL_CLR)
rds_info = "{}: {}: {}".format(artist, album, song)
radio.rds_buffer = rds_info.encode()

perc = float(status['elapsed']) / float(status['duration'])
draw.rectangle( (5, 215, 235, 230), outline=PROG_CLR)
draw.rectangle (
    (5, 215, 5 + int(230*perc), 230),
    fill=PROG_CLR
)

disp.image(image)

last_update = time.monotonic()

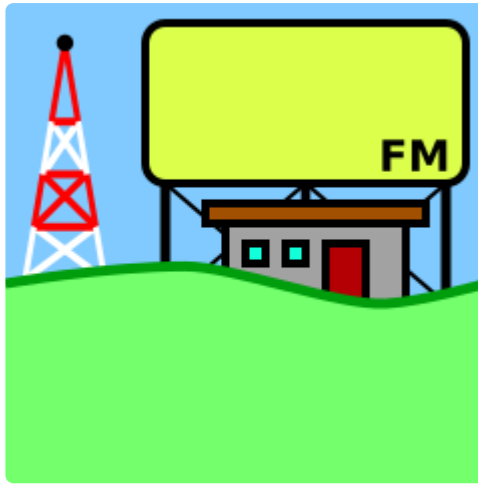
print("Now broadcasting {} on {}FM".format(STATION_NAME, FREQ))

while True:
    now = time.monotonic()
    try:
        status = mpc.status()
    except ConnectionError:
        mpc.connect("localhost", 6600)
        status = mpc.status()
    if not button1.value:
        button1_handler()
        while not button1.value:
            time.sleep(0.001)
    if not button2.value:
        button2_handler()
        while not button2.value:
            time.sleep(0.001)
    if now - last_update > UPDATE_RATE:
        update_display()
        last_update = now

```

Background Image

A static image file is used for the background on the PiTFT. Download this image:



And save it as **radio_bg.py** onto the Pi in **/home/pi** - the same directory where **radio.py** is saved.

Configure and Run

There are a few lines at the top of the **radio.py** code that can be changed. Look for these lines:

```
#--| User Config |-----  
FREQ = 89.00  
PLAYLIST = "test"  
STATION_NAME = "PiPyPirate Radio"  
UPDATE_RATE = 0.5  
#-----
```

And change, if needed, as follows:

- **FREQ** - Set this to the frequency that was found to be available from the frequency scan performed in the previous section.
- **PLAYLIST** - Set this to the playlist that will be broadcast over the radio. The default **test** playlist was created earlier in this guide.
- **STATION_NAME** - Change this text to be your station name. It actually gets broadcast (via RDS) and will show up on radios that display this kind of information.
- **UPDATE_RATE** - This sets how often the TFT display is refreshed, in seconds. The default 0.5 value should be fine, but the adjustment is here if needed.

Once the changes have been saved, run the **radio.py** program to start broadcasting:

```
python3 radio.py
```

It will print the station name and frequency being used. As long as the program is running, the radio is broadcasting.

```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ python3 radio.py  
Now broadcasting PiPyPirate Radio on 107.5FM
```

Info and Status

When the `radio.py` program runs, the PiTFT is updated with with playback information and status.



1. Animated radio "waves" will show around the antenna if the radio is broadcasting.
2. These three lines show artists, album, and title for the current song playing.
3. Along the bottom is a playback progress bar.
4. The billboard displays the currently set broadcast frequency. Tune to this!

Playback Control

The two buttons on the PiTFT provide some minimal control of playback.



1. Pause/resume playback. The radio "waves" will stop animating when paused.
2. Skip to next song in playlist.