ScoutMakes FM Radio Board STEMMA I2C

Created by Lalindra Jayatilleke

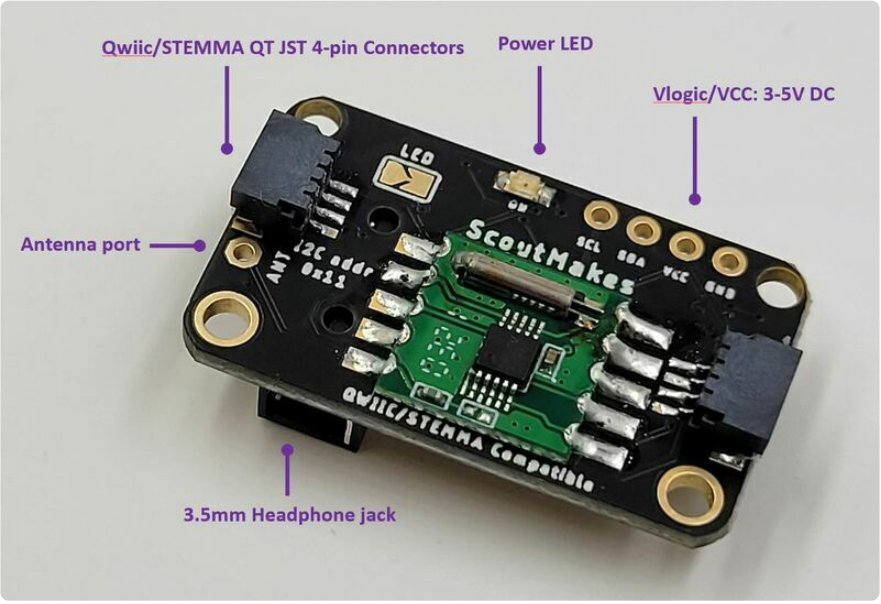
https://learn.adafruit.com/scoutmakes-fm-radio-board-stemma-i2c

Last updated on 2024-06-03 03:44:15 PM EDT

Table of Contents

Overview

The ScoutMakes FM radio board is a great way to learn about the world of electronics. It is a I2C breakout board for the RDA5807 FM radio chip by RDA Microelectronics. The board supports Arduino and CircuitPython. Libraries and example code are provided by to help you get started. The board also has connectors which support SparkFun's Qwiic and Adafruit's Stemma QT I2C connector standards, JST 4-pin.

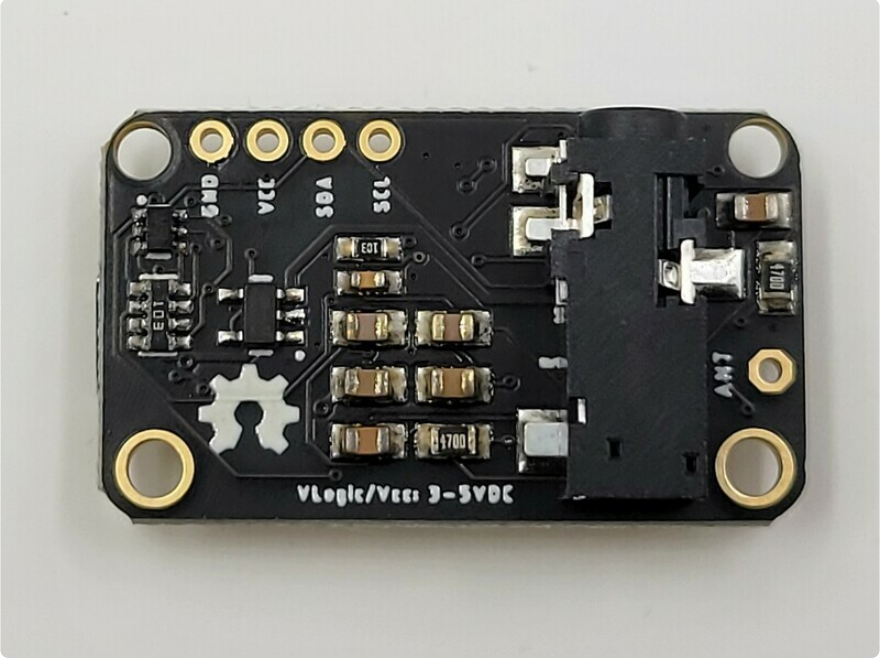


The RDA5807 also has RDS (Radio Data System) capability where station information (station name, song name) can be received. The example code shows RDS information as well.

Specifications:

- Qwiic/STEMMA QT compatible. JST SH 4-pin connectors.
- Runs on I2C standard.
- Supports Arduino and CircuitPython (library available in Adafruit's community bundle).
- 3-5V DC compatible.
- RDS data provided by example code.
- User menu provided in example code.
- ENIG RoHS compliant PCB.
- Power LED (trace can be cut to disable LED for power saving).
- 3.5mm Headphone jack.
- Open Source design.
- Antenna header for improved reception.

- Default I2C address 0x11.



Pinouts

The default I2C address is **0x11.**

Power Pins

- **VCC** - this is the power pin. We have included a voltage regulator on board that will take 3-5VDC and safely convert it down. To power the board, give it the

same power as the logic level of your microcontroller - e.g. for a 5V microcontroller like Arduino, use 5V.
- **GND** - common ground for power and logic.

I2C Logic Pins

- **SCL** - I2C clock pin, connect to your microcontroller I2C clock line. This pin is level shifted so you can use 3-5V logic, and there's a **10K pullup** on this pin.
- **SDA** - I2C data pin, connect to your microcontroller I2C data line. This pin is level shifted so you can use 3-5V logic, and there's a **10K pullup** on this pin.
- STEMMA QT **(https://adafru.it/Ft4)** - These connectors allow you to connectors to dev boards with **STEMMA QT** connectors or to other things with various associated accessories (https://adafru.it/JRA).

Power LED and LED Jumper

- **Power LED** - On the back of the board is the red power LED that would illuminate when there is power to the FM module. Labeled **on**.
- **LED jumper** - This jumper is located on the back of the board. Cut the trace on this jumper to cut power to the "on" LED.

Other

- **ANT** - This is the header where you can solder a piece of hook-up wire for an antenna.
- **3.5mm headphone jack** - This is the audio output of the FM board. You can connect a pair of headphones or an external speaker (https://adafru.it/18iD).

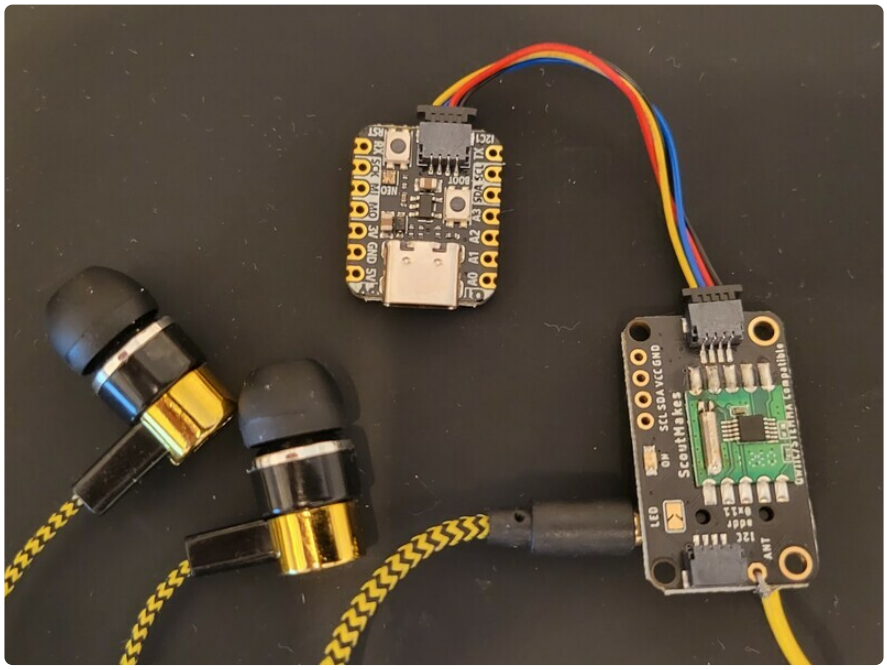
CircuitPython

It is very easy to get started using the ScoutMakes FM radio board with the RDA5807M radio module. There is a CircuitPython library along with example code in the Adafruit community bundle (https://adafru.it/ENC)

Solder a piece of wire onto the Antenna port on the breakout prior to testing. This will ensure good FM radio signal reception. The photos below show a twisted-wire connection.

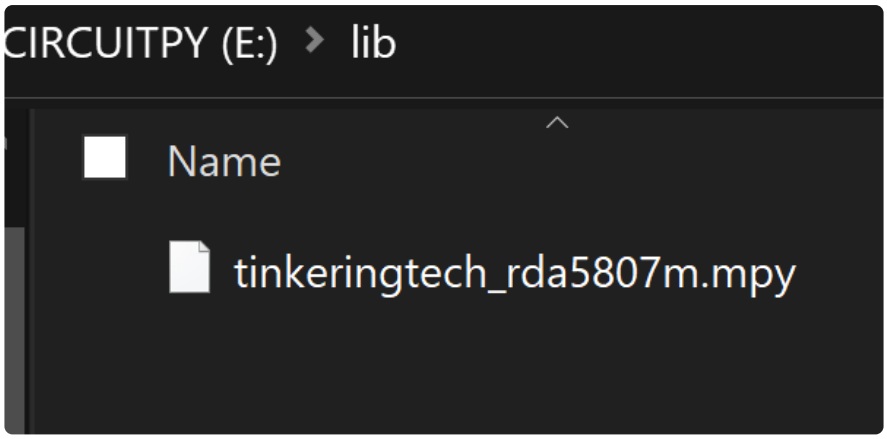
First you'll need to wire up the sensor for a basic I2C connection. The STEMMA QT connectors make it very easy to use the breakout with microcontroller board such as the Adafruit QT Py.

Connect the FM board with a STEMMA QT (http://adafru.it/4399) cable to the microcontroller. You will also need a pair of 3.5mm headphones or a connection to an external speaker (https://adafru.it/18iD) for sound output.



Connect a type-C USB cable to your QT Py board for power. The LED on the back of the FM board will light up indicating power on.

Place the FM breakout library **tinkeringtech_rda5807m.mpy** from the Adafruit community bundle (https://adafru.it/ENC) in your QT Py **lib** folder. You can find the library in the unzipped bundle in the **lib** folder as **tinkeringtech_rda5807m.mpy**.



An example CircuitPython sketch is located in the community bundle in the **examples** folder as **rda5807m_simpletest.py**. The latest version can also be found on GitHub (https://adafru.it/18ja).

The version as of December 2022 is reproduced below.

```
# SPDX-FileCopyrightText: Copyright (c) 2022 tinkeringtech for TinkeringTech LLC

import time
import board
import supervisor
from adafruit_bus_device.i2c_device import I2CDevice
import tinkeringtech_rda5807m

# Preset stations. 8930 means 89.3 MHz, etc.
presets = [8930, 9510, 9710, 9950, 10100, 10110, 10650]
i_sidx = 3  # Starting at station with index 3

# Initialize i2c bus
# If your board does not have STEMMA_I2C(), change as appropriate.
i2c = board.STEMMA_I2C()

# Receiver i2c communication
address = 0x11
vol = 3  # Default volume
band = "FM"

rds = tinkeringtech_rda5807m.RDSParser()

# Display initialization
toggle_frequency = (
    5  # Frequency at which the text changes between radio frequnecy and rds in
seconds
)
rdstext = "No rds data"

# RDS text handle
def textHandle(rdsText):
    global rdstext
    rdstext = rdsText
    print(rdsText)


rds.attach_text_callback(textHandle)

# Initialize the radio classes for use.
radio_i2c = I2CDevice(i2c, address)
radio = tinkeringtech_rda5807m.Radio(radio_i2c, rds, presets[i_sidx], vol)
```

```
radio.set_band(band)  # Minimum frequency - 87 Mhz, max - 108 Mhz

# Read input from serial
def serial_read():
    if supervisor.runtime.serial_bytes_available:
        command = input()
        command = command.split(" ")
        cmd = command[0]
        if cmd == "f":
            value = command[1]
            runSerialCommand(cmd, int(value))
        else:
            runSerialCommand(cmd)
        time.sleep(0.3)
        print("-&gt; ", end="")


def runSerialCommand(cmd, value=0):
    # Executes a command
    # Starts with a character, and optionally followed by an integer, if required
    global i_sidx
    if cmd == "?":
        print(
            """\
? help
+ increase volume
- decrease volume
&gt; next preset
&lt; previous preset
. scan up
, scan down
f direct frequency input; e.g., 99.50 MHz is f 9950, 101.10 MHz is f 10110
i station statuss mono/stereo mode
b bass boost
u mute/unmute
r get rssi data
e softreset chip
q stops the program"""
        )

    # Volume and audio control
    elif cmd == "+":
        v = radio.volume
        if v &lt; 15:
            radio.set_volume(v + 1)
    elif cmd == "-":
        v = radio.volume
        if v &gt; 0:
            radio.set_volume(v - 1)

    # Toggle mute mode
    elif cmd == "u":
        radio.set_mute(not radio.mute)
    # Toggle stereo mode
    elif cmd == "s":
        radio.set_mono(not radio.mono)
    # Toggle bass boost
    elif cmd == "b":
        radio.set_bass_boost(not radio.bass_boost)

    # Frequency control
    elif cmd == "&gt;":
        # Goes to the next preset station
        if i_sidx &lt; (len(presets) - 1):
            i_sidx = i_sidx + 1
            radio.set_freq(presets[i_sidx])
    elif cmd == "&lt;":
        # Goes to the previous preset station
        if i_sidx &gt; 0:
```

```
            i_sidx = i_sidx - 1
            radio.set_freq(presets[i_sidx])

    # Set frequency
    elif cmd == "f":
        radio.set_freq(value)

    # Seek up/down
    elif cmd == ".":
        radio.seek_up()
    elif cmd == ",":
        radio.seek_down()

    # Display current signal strength
    elif cmd == "r":
        print("RSSI:", radio.get_rssi())

    # Soft reset chip
    elif cmd == "e":
        radio.soft_reset()

    # Not in help
    elif cmd == "!":
        radio.term()

    elif cmd == "i":
        # Display chip info
        s = radio.format_freq()
        print("Station: ", s)
        print("Radio info:")
        print("RDS -&gt;", radio.rds)
        print("TUNED -&gt;", radio.tuned)
        print("STEREO -&gt;", not radio.mono)
        print("Audio info:")
        print("BASS -&gt;", radio.bass_boost)
        print("MUTE -&gt;", radio.mute)
        print("SOFTMUTE -&gt;", radio.soft_mute)
        print("VOLUME -&gt;", radio.volume)


print_rds = False
runSerialCommand("?", 0)

print("-&gt; ", end="")

while True:
    serial_read()
    radio.check_rds()
    new_time = time.monotonic()
    serial_read()
```

To run this example, we used the Mu Python code editor (https://adafru.it/ANO). If the code is running correctly on your board, you will see an output like below on your serial monitor in Mu.

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
? help
+ increase volume
- decrease volume
> next preset
< previous preset
. scan up
, scan down
f direct frequency input e.g 99.50 MHz is f 9950 or 101.10 MHz is f 10110
i station status
s mono/stereo mode
b bass boost
u mute/unmute
r get rssi data
e softreset chip
q stops the program
->                          Â¨Â¢Â Â
HOT995 D    ↔[@IT MUSICÂ¨Â¢Â Â TION
HOT995 D    ↔[@IT MUSIC STATION
HOT995 DC's  0 PIT MUSIC STATION
HOT995 DC's #1 HIT MUSIC STATION
```

The menu allows for control of the FM module. Just type in the desired command to view an output or send a particular setting command to the module. For example, if you want to increase the volume, type `+` and hit enter on the serial console

By default, the FM board will tune to FM 99.5 MHz. If this is not a suitable channel for your area, you can scan for other stations by entering the `>` or `<` command on the serial console to scan for available frequencies in your area.

Here is a great resource for finding FM radio stations in your area in the US FM radio channel locator (https://adafru.it/RE5)

If the FM radio station that you are tuned into has Radio Data System (RDS) (https://adafru.it/18jd), you will start to see the output of RDS on the serial monitor in real-time. You may notice garbage data initially as reception improves. If the radio signal is weak, you will see continuous garbage RDS output. Move to an area with better reception, or tune to a frequency that has a stronger signal for your area. You may see the signal strength by issuing a "`r`" command on the console.

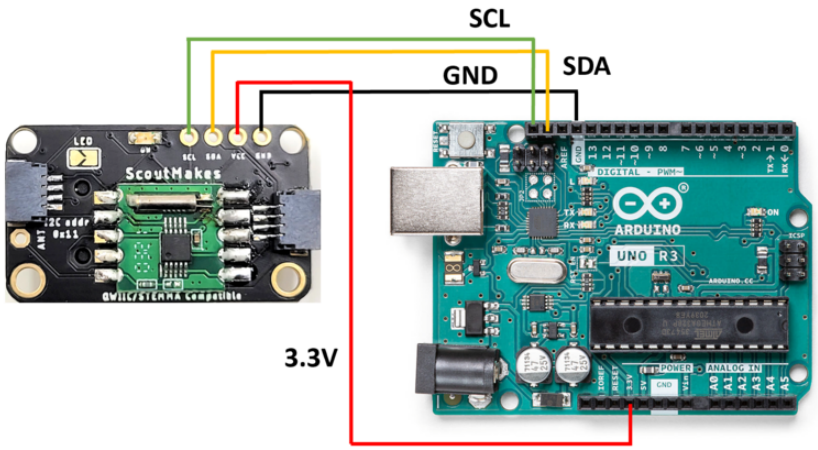
```
->                          Â¨Â¢Â Â
HOT995 D    ↔[@IT MUSICÂ¨Â¢Â Â TION
HOT995 D    ↔[@IT MUSIC STATION
HOT995 DC's  0 PIT MUSIC STATION
HOT995 DC's #1 HIT MUSIC STATION
```

Have fun playing with this awesome FM module!

Arduino

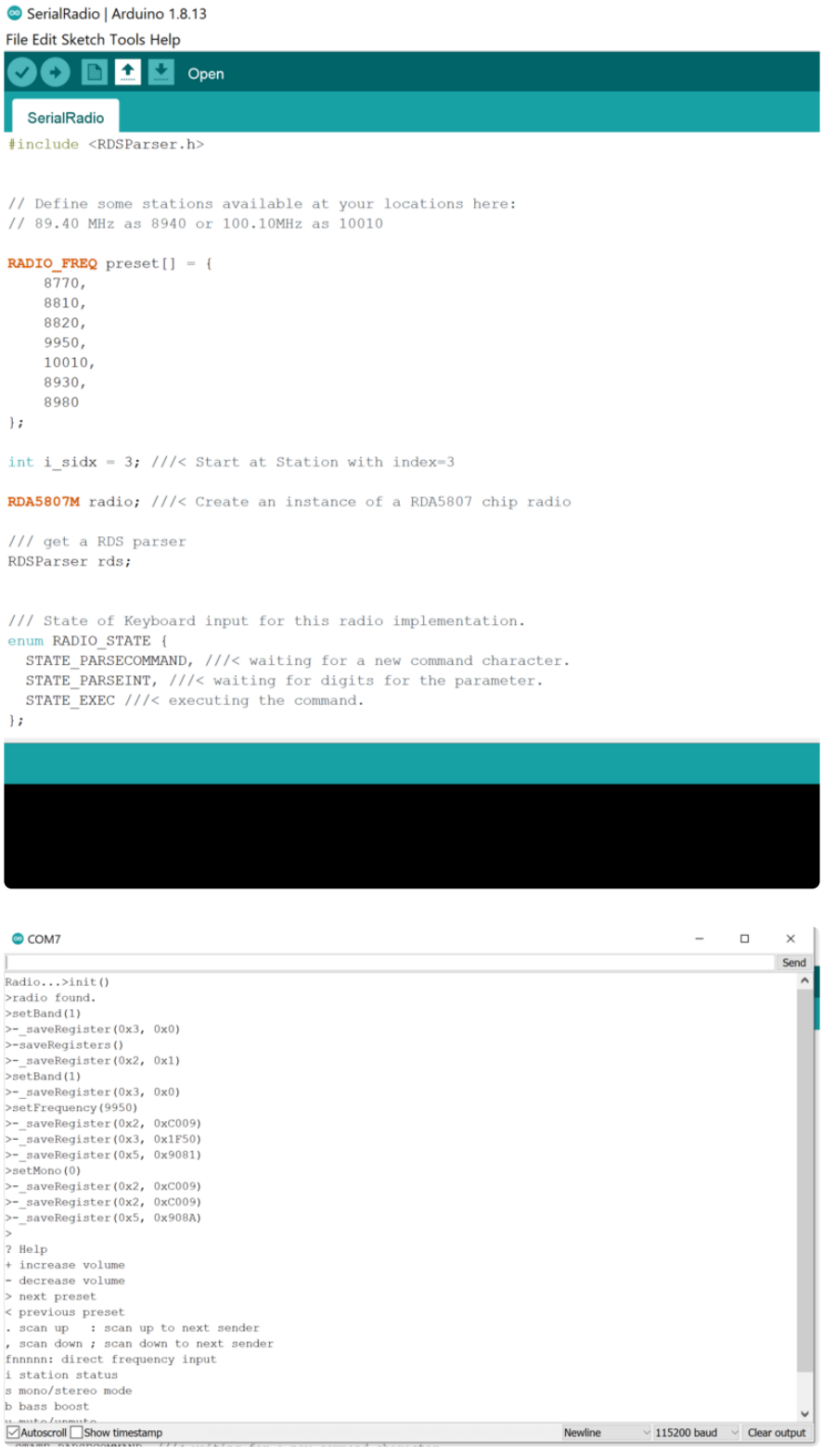
The ScoutMakes FM breakout board also has support for Arduino. We will use Matthias Hertel's RDA5807m library. The Arduino example and library can be found here (https://adafru.it/18je)

For this example we will use an Arduino UNO connected to the ScoutMakes FM board breakout pins for VCC, GND, SCL, SDA.

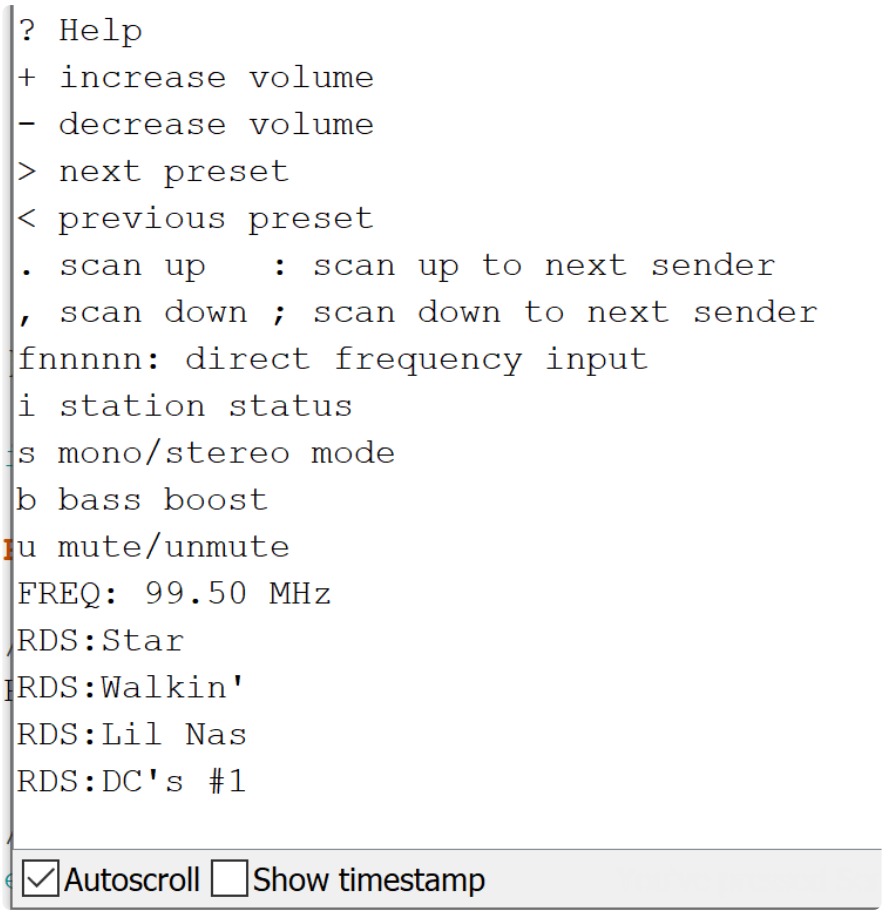


Once the breakout is connected to your Arduino, fire up your Arduino IDE and grab the example code and load SerialRadio.ino sketch. This example will enable you to test the module with a user menu and RDS data on Serial. The Arduino Serial console needs to run at 115200 baud.

Ensure you also have a pair of headphones or external speaker (https://adafru.it/18iD) connected to the 3.5mm headphone jack on the FM board along with an antenna wire for good reception.



Choose the various menu items to configure the radio. The Radio Data System (RDS) data will print out onto the serial console in real time.



Downloads

Schematics and board design files can be found here (https://adafru.it/18jA).