Richard J.S. Bates
wd1o@arrl.net WQZI552 December 2018

# An RPi3-based GMRS Radio Recorder System

## Summary

This note describes a headless Raspberry Pi-based message recorder system, designed to extend the capabilities of a community-based GMRS emergency radio. When unstaffed, the system forwards the first 10s of a recorded message as an email attachment, and makes the complete message securely accessible online. When staffed locally, the system provides additional message logging and annotating.

## Contents



## 1. Features

- The headless RPi3-based system records incoming radio messages and provides an audible receipt to the caller.
- The system can operate "stand-alone", or on a LAN, or with Internet access.
- "Stand-alone", the messages are replayed locally via an earphone.
- A LAN-connected web browser can play time-stamped messages and add notes.
- With Internet access, the first 10s of each message is emailed and the whole is securely accessible online.
- Emergency announcement messages can set up to be transmitted at regular intervals.

## 2. Introduction

This project arose from the need for an alternative means of emergency community communication, should conventional wireline and wireless telephony systems fail; for example, for people house-bound and/or with medical needs during a snow or ice storm. Issued with inexpensive FRS radios, community members may call the GMRS radio at the Town's Emergency Management Agency (EMA) for assistance. However with limited emergency personnel, there's a very practical possibility that no-one would be available in the radio room to immediately take the call; hence the need for a message recorder system.

The recorder system design covers three scenarios at the EMA radio room:

1. Just the radio and headless Raspberry Pi-based message recorder; unheard messages are indicated by a flashing light on the RPi case; all messages can be played back through an earphone.

2. Scenario #1 plus a networked local computer with a web browser to access the Raspberry Pi; all messages are time-stamped and can be accessed and annotated using the web browser.

3. Scenario #2 plus an live Internet connection; the first 10s of each message is sent as an email attachment and the complete message is accessible remotely via a cloud-based file sharing system.

Note that while the system in scenario #3 offers the most features, it's probably the least likely during a practical deployment; if wireline and wireless telephony is unavailable, the Internet is probably out as well.

What follows is a description of the hardware used for the project, notes on the construction, a description of the Python scripts, and finally some comments on the design choices.

In passing, the software philosophy for this project may be loosely labeled P4, as in "Pragmatic Plagiaristic Python Programing" - although, since the code ideas come from not one but countless examples on the web, perhaps the second P could be omitted[1], but it's definitely pragmatic; the code is not pretty but it works.

## 3. Hardware

The project used the following hardware – there are alternatives for many of the parts - the hyperlinks are to examples:

1. Midland MXT400 2-way radio
2. Raspberry Pi 3
3. Enokay case for Raspberry Pi
4. SanDisk Ultra 32 GByte MicroSD memory card
5. SYBA USB sound adapter
6. Push-button switch
7. Piezo buzzer
8. Ethernet extension cable
9. Mono mini-plug audio cable
10. Stereo mini-plug audio cable
11. Low-voltage reed relay
12. 5v supply – e.g. DROK 12v-5v converter
13. Earphone – e.g. classic
14. Green led indicator – e.g. 5mm led
15. 330Ω and 10kΩ 1/8W resistors

## 4. Construction

The assembly was simple, but by no means the only way to achieve the required functionality. The piezo buzzer (#7) and reed relay (#10) were mounted inside the RPi case (#3), and soldered directly to the I/O pins of the Raspberry Pi (#2); Table 1 lists the selected pins.

| I/O Connections | |
|---|---|
| Pin # | **Connection** |
| 29 | Green LED |
| 31 | PTT Relay for Microphone |
| 11 | Piezo Buzzer |
| 36 | Push-button Switch |

**Table 1 I/O Connections to the Raspberry Pi Bus**

[1] https://quoteinvestigator.com/2010/09/20/plagiarism/

The led indicator (#14) (and associated 330Ω resistor) and push-button switch (#6) were mounted in the end-plate of the case, as illustrated in Figure 1

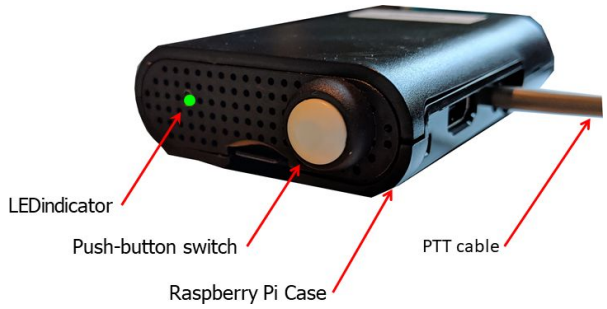



**Figure 1 Raspberry Pi Case**

The audio jack on the RPi circuit board was removed to make way for a 2-wire cable, labeled as the PTT cable, connecting the output of the reed relay to the PTT connection in the radio microphone cable. To avoid having to make any wiring changes to the Midland radio or microphone cable, an Ethernet extension cable (#8) was interposed between the microphone and the radio input; Figure 2 shows the various connections spliced into it.

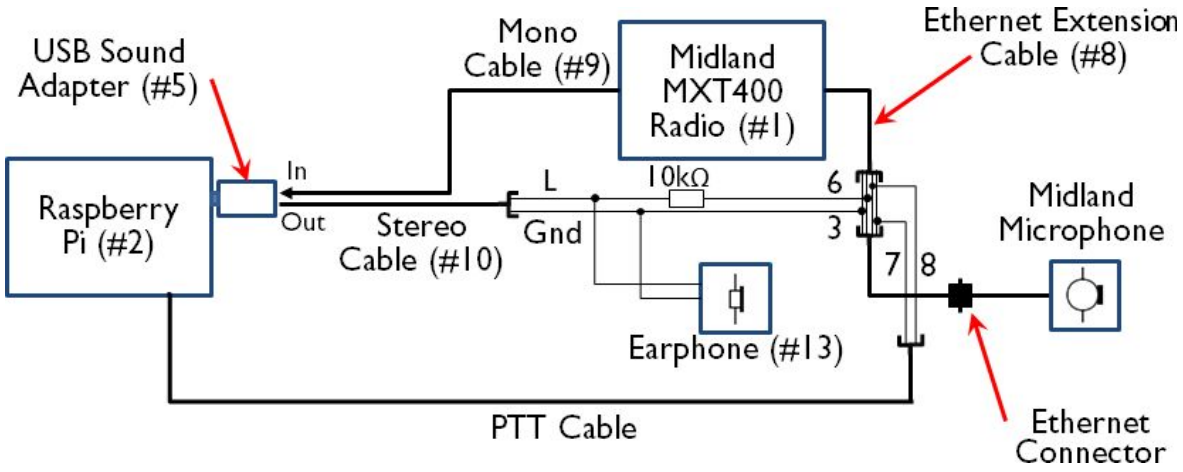
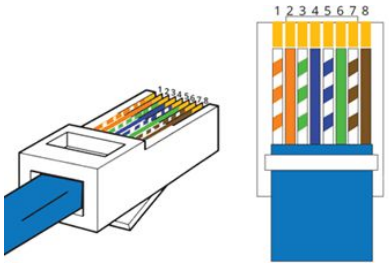



**Figure 2 Wiring Details for Audio and PTT Connections**

The numbers next to the spliced portion of the Ethernet extension cable refer to the connector wiring for the microphone that comes with the Midland MXT400 radio, as illustrated in Figure 3.




| Pin # | Function |
|---|---|
| 3 | Audio Ground |
| 6 | Audio Signal |
| 7 | PTT |
| 8 | PTT Ground |

**Figure 3 Wiring to connect with the Midland Microphone and PTT**

## 5. Software

There are two Python scripts to provide the main functionality:

1. `gmrs.py` - this records messages and provides a basic playback mechanism for scenario #1
2. `index.py` - this provides web browser access to playback and annotate messages for scenario #2

Scenario #3 is accomplished using parts of `gmrs.py`

Figure 4 shows the flowchart for the script `gmrs.py` to record and playback messages; the full script is listed in Appendix 2.

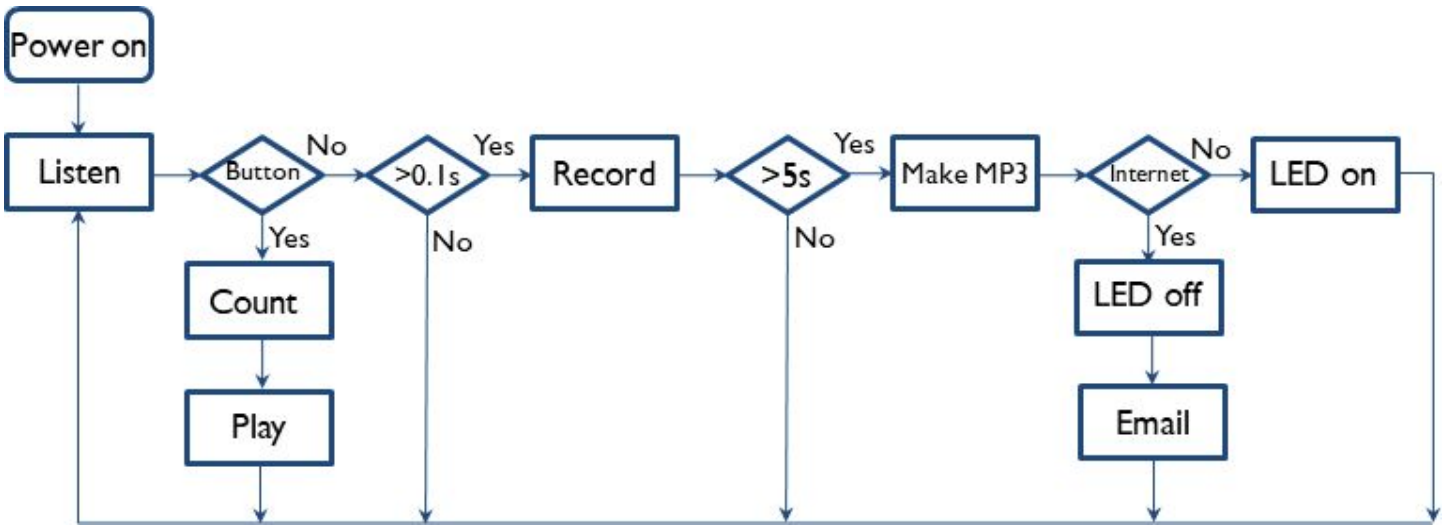



**Figure 4 Flowchart for Recorder Program gmrs.py**

In outline, the script listens for and records incoming messages, stores them as mp3 files, and emails the group. In more detail:

- The script normally loops around in the while statement (line 103), listening for incoming messages
- If the rms level of the 0.1s sample exceeds the noise threshold, the system starts recording (line 126)
- If the recorded sample is longer than 8 s (line 139), it is converted into an mp3 file, stored in the tonido file storage area `~/recordings/incoming` (line 171) and logged in the 'messages' table of the sqlite3 gmrsrecordingdata.db database (line 164)
- A test is made, to one of Google's DNS, to check the Internet is up (line 184)
- If up, an email with the first 10s of the message, is sent to the mailing list (line 190) and the complete message synchronized with the shared Google Drive folder (191) using rclone. An audible receipt is transmitted back to the caller, confirming that the message has been received (line 177).
- If down, the green LED is set to start flashing (line 105), to attract the attention of the local operator

- Stored messages can be heard through the earpiece; selecting a message to replay is done by holding in the white button (line 108); the buzzer will sound briefly at the end of every second; releasing the buzzer will then play the n'th most recent message (line 117), where n is the number of times the buzzer has sounded. For example, by holding in the button until the buzzer has sounded once, and then releasing, the system will play the most recent message.

The index.py script (and associated css file index.css) to handle scenario #2 is listed in Appendix 3; it reads (lines 28 & 61) and writes (line 24) data from the sqlite3 database gmrsrecorderdata.db and displays this in the browser. Figure 5 shows a screenshot of an example web browser output, where there is one pending received message, to be resolved, and two messages that have already been dealt with; messages, and their resolution are all time-stamped.

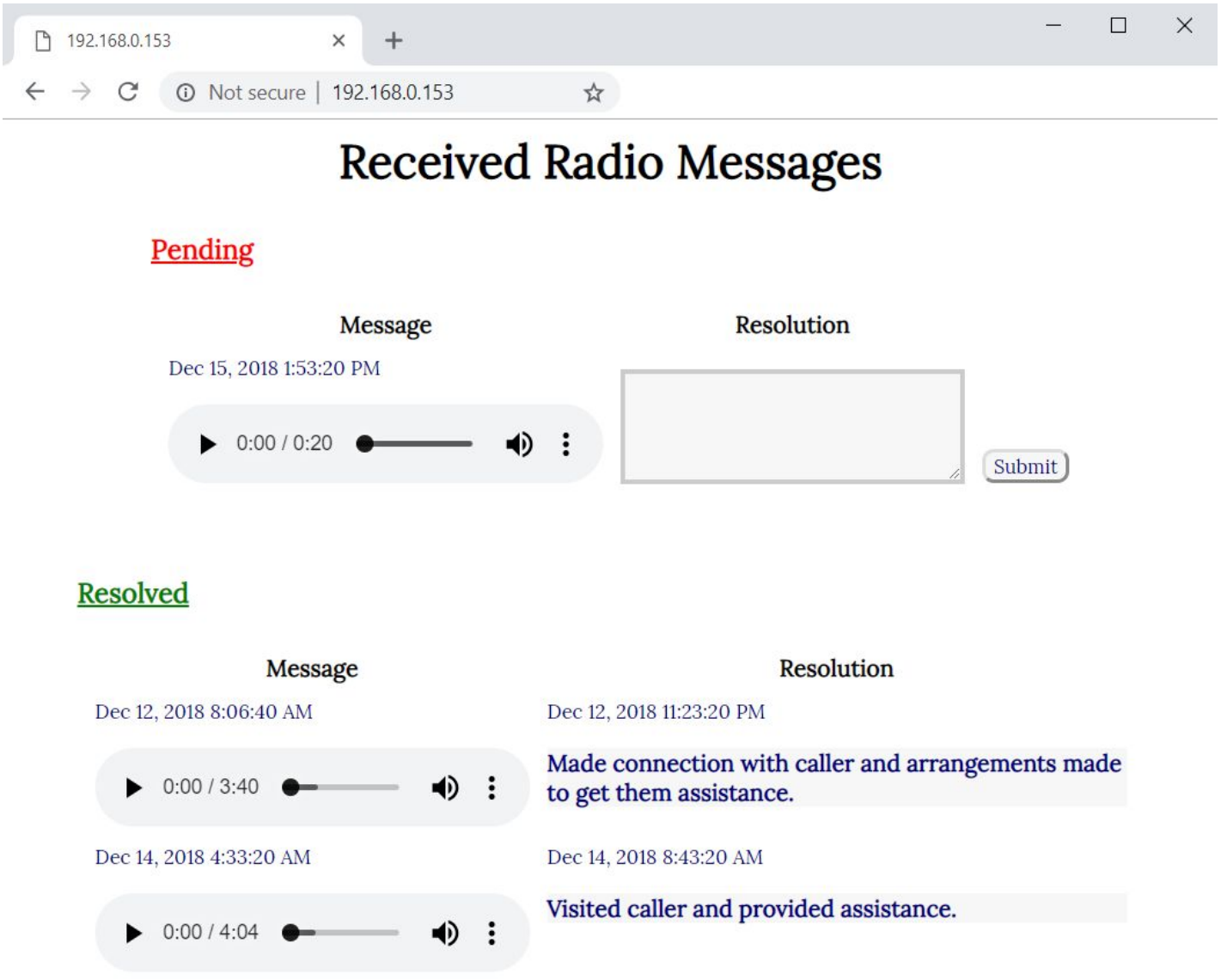



**Figure 5 Screenshot of Example Web Browser Output**

Figure 6 shows an example email notice of a recorded message that would be sent out in scenario #3, with Internet access. The recipient receives the first 10s of the message (line 190), together with a link to the whole message on Google Drive.

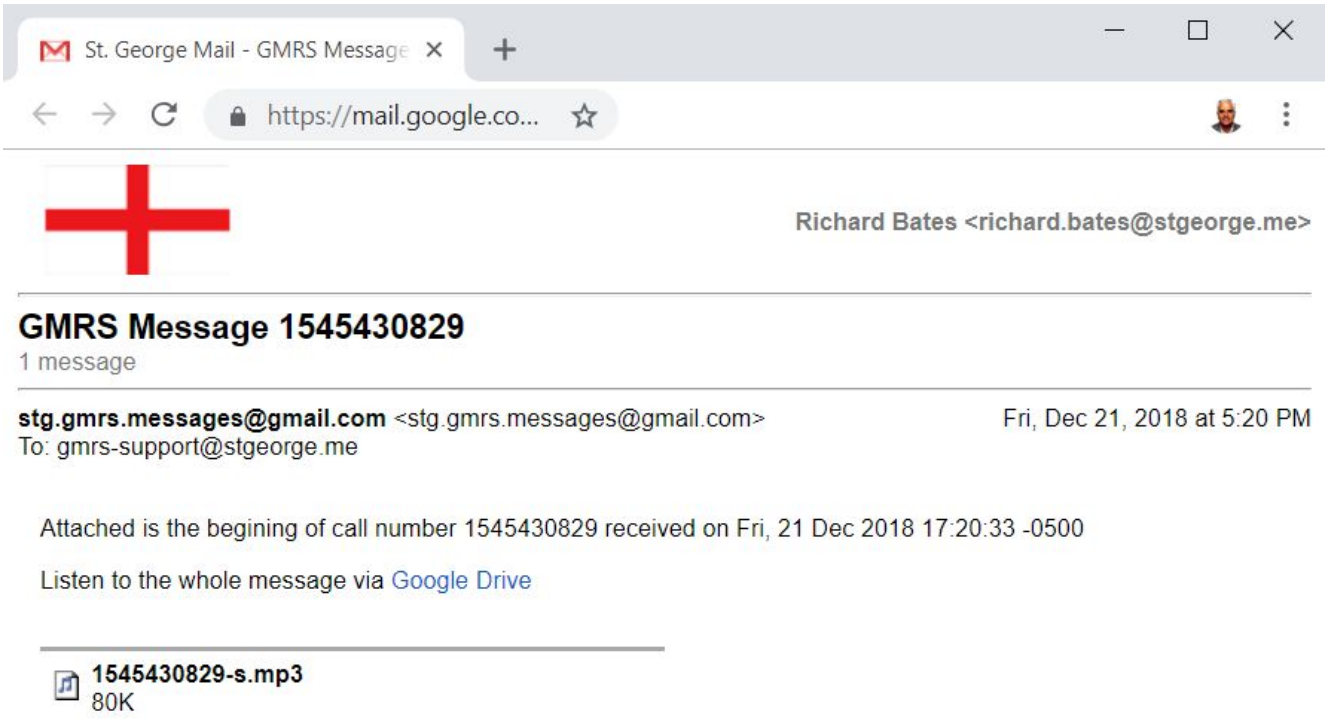



**Figure 6 Example Email Notice of Recorded Message**

## 6. Design Considerations

- The system was tweaked to avoid "false positives", i.e. to avoid labeling noise and interference as valid messages; this was done with CTCSS on the radio channel, by measuring and setting the noise threshold, and also with a minimum message duration (line 139).

- rclone can be set to synchronize the recorded messages with a large variety of cloud-based service; this design used the Google Drive service for personal convenience; other services like Dropbox, MS OneDrive, etc would be equally satisfactory.

- The design capability to email the first part of a message is enhanced by sending it to a mailing list, for example to a private Google Group; in this way a number of people can be simultaneously notified of the new call.

- An interesting addition to this project may be using Google's speech-to-text service, or similar, so that the received email includes a transcription of the messages.

# Appendix 1: Adjustments to Default Raspbian Setup

- The following packages were added to the default Raspbian Stretch operating system:

  python-pyaudio
  sox
  lame
  mpg123
  ntp
  sqlite3
  apache2
  rclone

- These notes were used to set the USB sound adapter as the default audio device.

- These notes were used to install and set up rclone, so that the folder `~/recordings/incoming` could be synchronized with a Google Drive folder; this Google Drive folder was set to be read-only shared with anyone who had the link.

- To have the `index.py` script execute in the `/var/www/html/host-cgi` directory on the Apache2 web server, the following detail was added to the `/etc/apache2/apache2.conf` configuration file

```
<Directory /var/www/html/host-cgi>
        Options +ExecCGI
        AddHandler cgi-script .py
</Directory>
```

  and the following added to the virtual host definition, to allow the server to access the recorded messages in the `~/recordings/incoming` directory

```
        ServerAdmin wd1o@arrl,net
        DocumentRoot /var/www/html/host-cgi
        DirectoryIndex index.py
        Alias /messages/ "/home/pi/recordings/incoming/"
        <Directory "/home/pi/recordings/incoming/">
                Order allow,deny
                Allow from all
                Require all granted
        </Directory>
```

  and, as described here, the cgi modules for Apache2 were enabled with the command

```
sudo a2enmod cgid
```

## Appendix 2: Recorder Script

Script gmrs.py to record and playback messages:

```
1          #!/usr/bin/python
2          # encoding: utf-8
3
4          import pyaudio
5          import audioop
6          import wave
7          import subprocess
8          import smtplib
9          import os
10         import glob
11         import datetime
12         import time
13         import sqlite3
14         import socket
15         import RPi.GPIO as GPIO
16         from email.mime.multipart import MIMEMultipart
17         from email.mime.base import MIMEBase
18         from email.mime.text import MIMEText
19         from email.utils import formatdate
20         from email import encoders
21         from shutil import copyfile
22
23         SAMPLE_TIME = 0.1  # in seconds
24         FORMAT = pyaudio.paInt16
25         CHANNELS = 1
26         RATE = 48000   # in Hz
27         END_OF_MESSAGE_SILENCE = 2.5 # in seconds
28         MAX_MESSAGE_TIME = 300  # in seconds
29         led=29    # i/o pin
30         buzzer=11   # i/o pin
31         micPTT=31   # i/o pin
32         switch=36   # i/o pin
33         duration = 2000   # beep length of 250 ms, with 4 kHz tone
34         internetUp = True  # start with Internet working
35         internetDownPast = False # start with no past issues
36         ledState = False  # start with green LED off
37
38         chunk = int(SAMPLE_TIME * RATE)
39         threshold_start = 0.6 * chunk
40         threshold_stop = 0.3 * chunk
41         eom_loop = int( END_OF_MESSAGE_SILENCE / SAMPLE_TIME)
42         maxm = int (MAX_MESSAGE_TIME / SAMPLE_TIME)
43
44         GPIO.setwarnings(False)
```

```
45          GPIO.setmode(GPIO.BOARD)
46          GPIO.setup(led, GPIO.OUT,initial=0)
47          GPIO.setup(buzzer, GPIO.OUT,initial=True)
48          GPIO.setup(switch, GPIO.IN, pull_up_down=GPIO.PUD_UP)
49
50          def beep(x,y):
51           buzzerState = False
52           while (x != 0):
53            GPIO.output(y,buzzerState)
54            buzzerState = not buzzerState
55            x -= 1
56            time.sleep(0.000045)
57
58          def send_call(x):
59           send_from = '<email-from-address>'
60           pw = '<email-from-password>'
61           send_to = '<email-to-address>'
62           outer = MIMEMultipart('alternative')
63           outer['Subject'] = 'GMRS Message ' + x
64           outer['From'] = send_from
65           outer['To'] = send_to
66           outer['Date'] = formatdate(localtime=True)
67           a = 'Attached is the beginning of call number ' + x + ' received on
' + outer['Date']
68           b = 'As the internet has been down earlier, check for past messages
that were not emailed.'
69           c = '</body></html>'
70           d =
'https://drive.google.com/drive/folders/1h7BsPowOqJtUq-L2whqpIQmtCtLDI8Li'
71           text = a + '\n\nListen to the whole message via Google Drive at ' +
d
72           html = '<html><head></head><body><p>' + a + '</p><p>Listen to the
whole message via <a href=\"' + d + '\">Google Drive</a></p>'
73           if internetDownPast:
74            text = text  + '\n\n' + b
75            html = html + '<p>' + b + '</p>'
76           html = html + c
77           part1 = MIMEText(text, 'plain')
78           part2 = MIMEText(html, 'html')
79           outer.attach(part1)
80           outer.attach(part2)
81           attachment = ['/home/pi/recordings/incoming-truncated/' + x +
'-s.mp3']
82           for file in attachment:
83            with open(file, 'rb') as fp:
84             msg = MIMEBase('application', "octet-stream")
85             msg.set_payload(fp.read())
86            encoders.encode_base64(msg)
87            msg.add_header('Content-Disposition', 'attachment',
filename=os.path.basename(file))
```

```
88           outer.attach(msg)
89          server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
90          server.ehlo()
91          server.login(send_from, pw)
92          server.sendmail(send_from, send_to, outer.as_string())
93          server.close()
94
95         while True:
96          p = pyaudio.PyAudio()
97          stream =
p.open(format=FORMAT,channels=CHANNELS,rate=RATE,input=True,frames_per_buffer=c
hunk)
98          r = 0
99          listenMessages = False
100         GPIO.output(led,False)
101         mp3Files = sorted((glob.glob("/home/pi/recordings/incoming/*.mp3")),
reverse=True)
102
103         while r < threshold_start:
104          r = audioop.rms(stream.read(chunk), 2)
105          if not internetUp:
106           GPIO.output(led,ledState)
107           ledState = not ledState
108          if ((not GPIO.input(switch)) and (len(mp3Files) > 0)):
109           listenMessages = True
110           time.sleep(0.5)
111           a = -1
112           while (not GPIO.input(switch)):
113            beep(duration,buzzer)
114            a = a + 1
115            time.sleep(0.75)
116           if (a >= 0):
117            os.system('mpg123 ' + mp3Files[min(a,(len(mp3Files)-1))])
118          if listenMessages:
119           break
120         x = 0
121         y = 0
122         frames = []
123         if not internetUp:
124          GPIO.output(led,True)
125
126         while ((r > threshold_stop) and (y < maxm)) :
127          data = stream.read(chunk,)
128          frames.append(data)
129          r = r + audioop.rms(data, 2)
130          x += 1
131          y += 1
132          if x == eom_loop:
133           r = r / eom_loop
134           x = 0
```

```
135         stream.stop_stream()
136         stream.close()
137         p.terminate()
138 
139         if (len(frames) > 80):
140 
141          xt = datetime.datetime.now().strftime("%s")
142          temp_wav_filename = xt + "-t.wav"
143          wf = wave.open(temp_wav_filename, 'wb')
144          wf.setnchannels(CHANNELS)
145          wf.setsampwidth(p.get_sample_size(FORMAT))
146          wf.setframerate(RATE)
147          wf.writeframes(b''.join(frames))
148          wf.close()
149 
150          cmd = 'sox ' + xt + '-t.wav ' + xt + '.wav rate 8000'
151          subprocess.call(cmd, shell=True)
152          wav = xt + '.wav'
153          cmd = 'lame --preset insane %s' % wav
154          subprocess.call(cmd, shell=True)
155 
156          cmd = 'sox ' + xt + '-t.wav ' + xt + '-s.wav rate 8000 fade 0 10 1'
157          subprocess.call(cmd, shell=True)
158          wav = xt + '-s.wav'
159          cmd = 'lame --preset insane %s' % wav
160          subprocess.call(cmd, shell=True)
161 
162          conn=sqlite3.connect('/home/pi/gmrs/db/gmrsrecordingdata.db')
163          curs=conn.cursor()
164          y = "INSERT INTO messages values(null,\'" + xt + "\','0','','1')"
165          curs.execute(y)
166          conn.commit()
167          conn.close()
168 
169          cmd = 'mv ' + xt + '-s.mp3 ~/recordings/incoming-truncated/' + xt +
'-s.mp3'
170          subprocess.call(cmd, shell=True)
171          cmd = 'mv ' + xt + '.mp3 ~/recordings/incoming/' + xt + '.mp3'
172          subprocess.call(cmd, shell=True)
173          cmd = 'rm ' + xt + '*'
174          subprocess.call(cmd, shell=True)
175 
176          GPIO.setup(micPTT, GPIO.OUT,initial=1)
177          os.system('mpg123
~/recordings/outgoing/thank-you-message-received.mp3')
178          GPIO.output(micPTT,GPIO.LOW)
179 
180          internetUp = True
181 
182          try:
```

```
183           socket.setdefaulttimeout(3)
184           socket.socket(socket.AF_INET,
socket.SOCK_STREAM).connect(("8.8.8.8", 53))
185          except:
186           internetUp = False
187           internetDownPast = True
188
189          if internetUp:
190           send_call(xt)
191           cmd = 'rclone sync /home/pi/recordings/incoming
remote:GMRS/recorder/incoming'
192           subprocess.call(cmd, shell=True)
193           internetDownPast = False
194
195          print "file saved"
```

This script is set to run as a service at boot using the following steps:

1. Create the file `run_gmrs` in `~/gmrs`

```
cd /home/pi/gmrs
sudo -u pi ./gmrs.py
```

2. Create file `/etc/systemd/system/gmrs.service` containing:

```
[Unit]
After=network.target

[Service]
ExecStart=/bin/bash /home/pi/gmrs/run_gmrs
Restart=always

[Install]
WantedBy=multi-user.target
```

3. Run command:

```
sudo systemctl enable gmrs.service
```

## Appendix 3: Received Radio Messages Script

Script index.py to playback messages on LAN-connected web browsers and record resolution

```
1           #!/usr/bin/env python
2
3           import time;
4           import datetime;
5           import cgi;
6           import sqlite3;
7           import cgitb;cgitb.enable();
8
9           conn=sqlite3.connect('/home/pi/gmrs/db/gmrsrecordingdata.db')
10          curs=conn.cursor()
11          form=cgi.FieldStorage()
12
13          print 'Content-Type: text/html\n\n'
14          print '<html>\n<head>\n<link rel="stylesheet" type="text/css"
href="http://fonts.googleapis.com/css?family=Lora">'
15          print '<link rel="stylesheet" href="index.css"></head>\n<body>'
16          print ' <h1>Received Radio Messages</h1>'
17
18          for row in curs.execute("SELECT * FROM messages"):
19                  v =  "resolution" + str(row[0])
20                  if v in form:
21                          w = str(row[0])
22                          x = str(int(time.time()))
23                          y = form.getlist(v)
24                          z = 'UPDATE messages SET pending="0", resolved=\"' + x + '\",
resolution=\"' + y[0] + '\" WHERE id=\"' + w + '\"'
25                          curs.execute(z)
26          conn.commit()
27
28          curs.execute("SELECT * FROM messages WHERE pending='1'")
29          row = curs.fetchone()
30          if row is None:
31                  print '<h3 id="p-c">There are no unresolved messages</h3>'
32          else:
33                  print '<form action="index.py" method="POST" id="usrform">'
34                  print ' <center>'
35                  print '  <table cellpadding="6" cellspacing="0">'
36                  print '   <tr><td colspan="4"><h3 id="p-l">Pending</h3></td></tr>'
37                  print '   <tr>'
38                  print '    <td><p><br/></p></td>'
39                  print '    <td><p id="cen"><b>Message</b></p></td>'
40                  print '    <td><p id="cen"><b>Resolution</b></p></td>'
41                  print '    <td><p><br/></p></td>'
42                  print '   </tr>'
43                  for row in curs.execute("SELECT * FROM messages WHERE pending='1'"):
44                          print '   <tr valign="bottom">'
45                          print '    <td><p><br/></p></td>'
46                          x = datetime.datetime.fromtimestamp(row[1]).strftime("%b %d, %Y
%-I:%M:%S %p")
47                          print '    <td><p id="s-t">' + x  + '</p><p>'
```

```
48                          x = 'src=\"messages/' + str(row[1]) + '.mp3\"'
49                          print '     <audio controls><source ' + x + '
type="audio/mp3"></audio></p></td>'
50                          print '    <td><p>'
51                          x = 'name=\"resolution' + str(row[0]) + '\"'
52                          print '     <textarea ',x,' form="usrform"
value=""></textarea></p></td>'
53                          print '    <td ><p>'
54                          print '     <input type="submit" value="Submit"
id="but"></p></td>'
55                          print '   </tr>'
56                  print '  </table>'
57                  print ' </center>\n</form>'
58
59          print ' <p><br/></p>'
60
61          curs.execute("SELECT * FROM messages WHERE pending='0'")
62          row = curs.fetchone()
63          if row is None:
64                  print '<h3 ids="r-c>There are no resolved messages</h3>'
65          else:
66                  print ' <center>'
67                  print '  <table cellpadding="6" cellspacing="0">'
68                  print '   <tr><td colspan="3"><h3 id="r-l">Resolved</h3></td></tr>'
69                  print '   <tr>'
70                  print '    <td><p><br/></p></td>'
71                  print '    <td><p align="center"><b>Message</b></p></td>'
72                  print '    <td><p align="center"><b>Resolution</b></p></td>'
73                  print '    <td><p><br/></p></td>'
74                  print '   </tr>'
75                  for row in curs.execute("SELECT * FROM messages WHERE pending='0'"):
76                          print '   <tr valign="top">'
77                          print '    <td><p><br/></p></td>'
78                          x = datetime.datetime.fromtimestamp(row[1]).strftime("%b %d, %Y
%-I:%M:%S %p")
79                          print '    <td><p id="s-t">' + x  + '</p><p>'
80                          x = 'src=\"messages/' + str(row[1]) + '.mp3\"'
81                          print '     <audio controls><source ' + x + '
type="audio/mp3"></audio></p></td>'
82                          x = datetime.datetime.fromtimestamp(row[2]).strftime("%b %d, %Y
%-I:%M:%S %p")
83                          print '    <td width="400"><p id="s-t">' + x  + '</p><p>'
84                          x = str(row[3])
85                          print '     <p id="res">' + x + '</p></td>'
86                          print '   </tr>'
87                  print '  </table>'
88                  print ' </center>'
89
90          conn.close()
91          print '\n</body>\n</html>'
```

Richard J.S. Bates
wd1o@arrl.net WQZI552 December 2018

and the associated css file index.css

```
1          body { font-family: Lora; }
2          h1 { text-align: center; }
3          h3 { text-align: center; text-decoration: underline; }
4          td { background: transparent; border: none; padding: 5px; }
5          textarea { height: 4em; width: 25em; border: 3px solid #CCCCCC;
background-color: #F7F7F7;  color: navy; font-weight: bold; }
6          #cen { text-align: center; }
7          #res { padding: 5px; font-weight: bold; color: navy;
background-color: #F7F7F7; }
8          #but { border-radius: 8px; color: navy; background-color: #F7F7F7; }
9          #p-l { color: red; text-align: left; }
10         #r-l { color: green; text-align: left; }
11         #p-c { color: red; text-align: center; }
12         #r-c { color: green; text-align: center; }
13         #s-t { text-align: left; color: navy; font-size: 85%; }
```

## Appendix 4: Announcement Script

Script `announcement.py` that could be used to make announcements during emergencies, as permitted by the FCC Part 95 Personal Radio Services Rules.

The script could be launched by a crontab entry, for example

```
0 * * * * /home/pi/gmrs/announcement.py
```

so that it runs every hour, on the hour, transmitting the audio file `WQZI552-announcement.mp3`:

```
1       #!/usr/bin/python
2       import  RPi.GPIO  as  GPIO
3       import  time
4       import  os
5       GPIO.setwarnings(False)
6       GPIO.setmode(GPIO.BOARD)
7       micPTT=31
8       GPIO.setup(micPTT,  GPIO.OUT,initial=1)
9       os.system('mpg123  ~/recordings/outgoing/WQZI552-announcement.mp3')
10       GPIO.output(micPTT,GPIO.LOW)
```