

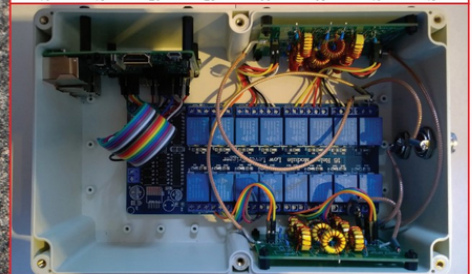
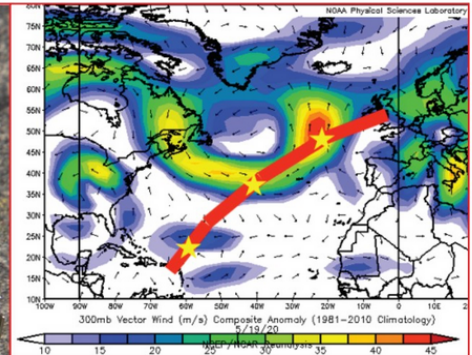
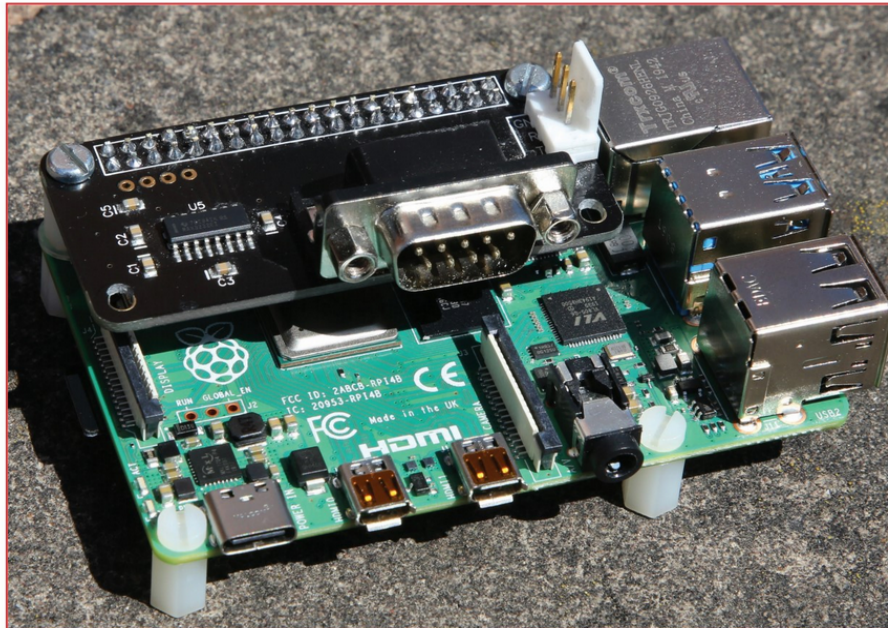


RADIO SOCIETY OF GREAT BRITAIN
ADVANCING AMATEUR RADIO SINCE 1913

RadCom

plus

SPRING 2021
VOLUME 06 ♦ Number 01 ♦ £4.95



RadCom plus

THE RADIO SOCIETY OF GREAT
BRITAIN'S MEMBERS' MAGAZINE

EDITOR:

Peter Duffett-Smith, G3XJE
E-mail radcomplus@rsgb.org.uk

TECHNICAL EDITOR:

Giles Read, G1MFG
E-mail giles.read@rsgb.org.uk

All contributions and correspondence concerning the content of *RadCom Plus* should be sent to: The Editor, *RadCom*, 3 Abbey Court, Fraser Road, Priory Business Park, Bedford MK44 3WH telephone 01234 832 700 Facsimile 01234 831 496, e-mail radcom@rsgb.org.uk

Notices to readers concerning errors and omissions and advertisements can be found at www.rsgb.org/radcomplus

RadCom Plus is published by the Radio Society of Great Britain as an addition to its official journal, *RadCom*, and is available free to all Members of the Society via the RSGB website at www.rsgb.org

All material in *RadCom Plus* is subject to editing for length, clarity, style, punctuation, grammar, legality and taste. No responsibility can be assumed for the return of unsolicited material (if in doubt, call us first!)

© Radio Society of Great Britain 2021. Articles are accepted on the strict understanding that they are not currently on offer to any other publication. Unless otherwise indicated the RSGB has purchased all rights to published articles.

Welcome to the May 2021 edition of *RadCom Plus*, the online journal for the technically-minded radio amateur. We have five articles in this edition. In the first of two contributions, Bob Cowdery, G3UKB describes how to operate relays remotely over a network, and in the second he applies that to the particular case of automating a Sotabeam WSPRLite antenna tester. Then we have a comprehensive and fascinating review of the mechanisms causing Sporadic-E propagation by Jim Bacon, G3YLA, shedding light on what, for me, has always been a somewhat mysterious mode of propagation. Continuing the theme of doing things remotely, Ian Sumner, G3VPX describes his piWebCat system that allows you to connect your rig to a network and operate it from anywhere. Both Bob's and Ian's systems are based on the Raspberry Pi computer. The fifth article is by Alan O'Donovan, G8NKM, and it describes his project to develop a wireless temperature sensor. Although not strictly amateur radio, I think that this is a well-written piece describing how he went about achieving a technical goal, solving issues similar to those we face in radio projects. I hope you enjoy reading it as much as I did.

I am beginning to collect articles for the second 2021 edition of *RadCom Plus*, which, I hope, will appear in November, but I need more! Please think about whether what you have been doing would be of interest to others, and get in contact with me. Or, if you don't think you could contribute yourself but would nevertheless like to see something on a particular subject, send me an email. I will be delighted to hear from you.

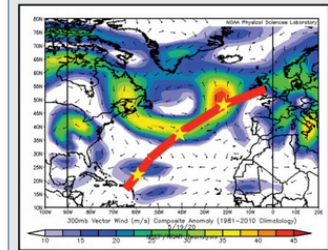
Peter Duffett-Smith G3XJE

Technical Features

- 3 Driving relays over a network
Bob Cowdery, G3UKB
- 8 An Automation system for the WSPRLite
Bob Cowdery, G3UKB
- 13 Sporadic-E – where we are now?
Jim Bacon, G3YLA
- 26 piWebCAT: a CAT software system for transceiver control
Ian Sumner, G3VPX
- 30 Developing a Wireless Temperature Sensor
Alan O'Donovan, G8NKM



COVER IMAGE 1: Page 26: piWebCAT



COVER IMAGE 2: Page 13: Sporadic-E, where are we now?



COVER IMAGE 3: Page 3: Driving relays over a network

Driving relays over a network

A relay might be the simplest of electro-mechanical devices, but there is no denying its usefulness. This article explains in some detail the software involved in driving relays over your home network, and illustrates three example projects that use this software: a 5V multi-output IP switch for power control of up to 8 devices using an inexpensive 8-relay board (**Photo 1**), a 4-way port switch using a good quality 4-relay board (**Photo 2**), and a low-pass filter selector using an inexpensive 16-relay board together with two SOTABEAMS 3-way low-pass filter boards (**Photo 3**, and described in more detail in the next article in this issue of RadCom Plus).

The software described here is a web application. This means it runs in a web browser on practically every device from a phone to a desktop. It is written in the popular Python programming language (plus a little javascript and css), which is both friendly for beginners as well as being powerful enough for seasoned coders. This article cannot teach you Python. If you are not familiar with Python, and perhaps have not even programmed before, then work through some tutorials first, although of course the software can be used as it is without needing to understand the code.

The target device on which the web server software runs at the remote end is a RaspberryPi (RPI).

GETTING THE SOFTWARE

Download the RPiWebRelay software from **[1]**. (The references are provided at the end of this article.) You can either go to the link and download a zip file, or use git to clone the repository. You can download the software onto your main machine or directly onto the target RPI. (Note that the RPI runs under a Unix type of operating system, and I have made the assumption your main machine does likewise.) Either way, the easiest method is to create a directory of your choice then:

```
$ cd <your directory>
$ git clone <URL>
```

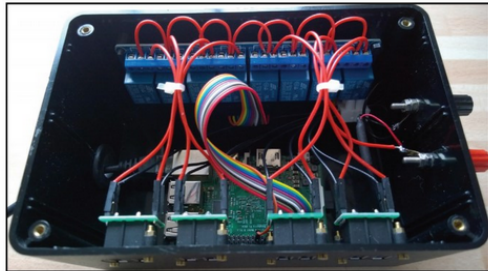


PHOTO 1: 8-port 5V switch.



PHOTO 2: 4-way port switch.

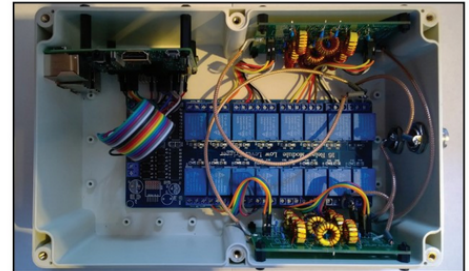


PHOTO 3: 6-band LPF filter selector.

(Here, the triangle braces denote a substitution. Thus, if your directory name is "remote_relays", then substitute "remote_relays" for "<your directory>", so that the command line becomes `$ cd remote_relays.`) You may need to run `apt-get install git` on your RPI first. The RpiWebRelay software will run on your PC. It won't be able to flip relays of course but it will recognise that it is not running on an RPI and instead will print out what the relay state is on each change. This is useful for testing.

PREPARING THE HARDWARE

This is not an RPI tutorial, so I have assumed you have a model 2, 3 or 4 with the latest RaspberryPi OS installed. The application is pretty lightweight, so a model 2 will do just fine. Almost everything you need will already be installed with the latest Raspberry Pi OS except the Python Web framework used by the application. This is called CherryPy **[3]** and is easy to install. You may first need to install pip **[4]**:

```
$ sudo apt-get install python3-pip
$ pip3 install cherrypy
```

Note that the software uses Python 3, hence you must use pip3 and not pip.

DRIVING THE RELAYS

The RPI has a number of interface pins on a 40-pin header. Fourteen provide the General Purpose Input/Output (GPIO) pins; other pins are designated ground, 5V, 3.3V and various other input/output

Bob Cowdery, G3UKB
bob@bobcowdery.plus.com

(I/O) functions. This means you can drive up to 14 relays without using an expansion board. The pin designations are not in numerical order, so ultimately some mapping is required between a pin number and a relay number as we are only interested in which relay is activated and not in which pin is used to activate it.

To drive a relay, we need a relay driver connected to a digital I/O pin which can then energise the relay when the pin is driven high (or low in some cases). Although building your own relay board with drivers would not be too hard, there are many boards available off the shelf (see e.g. [5] and [6]). In particular, there are low-cost boards that come in 5V or 12V versions with 4, 8 or 16 relays.

Fortunately, the Python installed on the RPi already has a library for GPIO control, so you don't need to install anything extra. In the RpiWebRelay download, the file called `webrelay_gpio.py` is the low-level relay driver re.g.ardless of which commercial or home-brew relay board you use.

Let's walk through this module to see the principles involved. (Lines starting with the `#` symbol are comments.)

```
# System imports
import os, sys

# Library imports
testing = False
try:
    import RPi.GPIO as GPIO
except ModuleNotFoundError:
    print("Error importing RPi.GPIO! Using test mode.")
    testing = True
```

The import statement loads extra modules. The modules named `os` and `sys` are standard modules that are often required, so I just import them as a matter of course. The `RPi.GPIO` module is the GPIO library which is imported and given the short-form name "GPIO", so it can be referred to using this short-form name subsequently in the program. The `try/except` structure around the import statement is a standard way to catch errors, which are called exceptions. The reason for doing this is so that the module can be run on any computer for testing as previously mentioned.

The code in this section of the program has been simplified a little to aid understanding. Note that `"_ "` represents two consecutive `"_ "` symbols.

```
#####
# The main GPIO class
#####
class GPIOControl:

    def __init__(self, num_relays, pin_map, inverse):
```

```
self.__num_relays = num_relays
self.__pin_map = pin_map
self.__inverse = inverse
```

if not testing:

```
# Set to use actual port numbering rather than pin numbering
GPIO.setmode(GPIO.BCM)
# Set all to output and state off (note inverted logic) as we are driving relays
for relay in range(0,num_relays):
    pin = self.__pin_for_relay(pin_map, relay)
    GPIO.setup(pin, GPIO.OUT)
    if inverse:
        GPIO.output(pin, GPIO.HIGH)
    else:
        GPIO.output(pin, GPIO.LOW)
```

Python is an object-oriented language. This is not something you need to understand in order to understand the code. If this is a new concept, do a web search; there are plenty of resources.

When a class is instantiated, which just means creating an instance of the code so it can be used in subsequent calls, Python automatically calls a method with the signature `__init__()`. In our case this call takes some parameters:

num_relays: this is the number of relays we have connected to the GPIO.

pin_map: as previously mentioned, we need to know which pin energises which relay. Thus, if our relays are numbered, say, 1 to 8 and we need to know which pin is connected to which relay, our pin map might look something like this: [2,3,4,17,27,22,10,9]. This is called a Python array and is indexed starting at zero, so the elements are 0 to 7 in this case. Indexing starting at zero is a common cause of errors! We refer to the relays being 1 to 8 and not 0 to 7, so keep this in mind as we go through the code.

inverse: some boards are active-low rather than active-high. This means that the relay is energised when a GPIO pin is in the low state, so we need to take this into account. If this flag is true, then the board is active-low.

Using these three pieces of information, we can initialise our GPIO pins unless we are in test mode when we skip this part:

```
GPIO.setmode(GPIO.BCM)
```

There are two pin numbering schemes employed by the library. The first, called `GPIO.BOARD`, uses the header pin numbering scheme. The second, called `GPIO.BCM`, uses the chip numbering scheme, ie the GPIO pin numbers you see on the header diagrams. I find this scheme easier to

follow.

```
for relay in range(0,num_relays):
    pin = self.__pin_for_relay(pin_map, relay)
    GPIO.setup(pin, GPIO.OUT)
    if inverse:
        GPIO.output(pin, GPIO.HIGH)
    else:
        GPIO.output(pin, GPIO.LOW)
```

We now loop from 0 to num_relays to initialise each pin. A pin can be an input or an output; clearly for a relay we want this to be an output (GPIO.OUT). We then set the pin to the de-energised state for the relay and this depends on whether the board is active-high or active-low.

Note that the line

```
pin = self.__pin_for_relay(pin_map, relay)
```

carries out a simple lookup of the pin number for this relay using the pin map.

This completes the initialisation. All we need now is a way to change the relay state:

```
# Set the relay to the given state
def set_relay(self, relay, state):
    pin = self.__pin_for_relay(self.__pin_map, relay)
    if testing:
        print("Setting pin %d to state %s" % (pin, state))
    else:
        if state == 'on':
            if self.__inverse:
                GPIO.output(pin, GPIO.LOW)
            else:
                GPIO.output(pin, GPIO.HIGH)
        else:
            if self.__inverse:
                GPIO.output(pin, GPIO.HIGH)
            else:
                GPIO.output(pin, GPIO.LOW)
```

The `set_relay` method takes two parameters:

relay: this is the relay number. It is time to state that this module works with relay numbers starting at zero. The user interface has the responsibility to change to relay numbers starting at one.

state: I chose to use the string 'on' for when the relay is energised, and 'off' for when it is not energised.

As described above, we first get the actual GPIO pin. Then, if we are testing, we simply print what we would do, otherwise we set the pin high or low depending on the **inverse** flag.

USER INTERFACE

Having a module that can drive relays given the right sequence of calls gives us a base for writing a user interface (UI). Now this can take many forms. It could be a tightly-coupled desktop UI written in one of the so-called GUI toolkits of which there is a good choice in Python. By "tightly-coupled" I mean that the UI has to be run on the same RPi as the relay driver. This might be what is required, but it is more useful to be able to run the UI on a different machine on the network. There are two options for doing that, a desktop UI that talks to the relay driver module over the network, or a web application that can be accessed from any machine that has a browser. The first requires a program that runs on the client machine, the second just requires a URL to be typed into your browser. A program also has to be started on the RPi in both cases but we will come to that later.

For this application, I decided to implement a web interface for its client-side simplicity. In fact I've implemented two interfaces. The first one is simplistic and is intended for automation, although it can still be run from your browser. This is the one I will describe here. The second is also a full web interface (included in the download) but the code is more complex and it is beyond the scope of this article to explain. However, the simple interface does demonstrate how straight forward a web interface can be, thanks to the Python web framework, CherryPy.

The CherryPy framework really elevates the code you need to write to the level where it looks pretty much like any Python module. CherryPy is well documented with a good tutorial section. The minimal user interface code can be found in the file `webrelay_min.py`

```
# System imports
import os, sys
import json
```

```
# Library imports
import cherrypy
```

```
# Application imports
import webrelay_gpio
```

As before, we have the import statements. In this case we import json, a simple data format. Its only purpose here is to read in a simple configuration file that contains the parameters that our GPIO module needs. In this way we can accommodate different relay mappings without changing any of the code. We also import the web framework cherrypy and our webrelay_gpio module as previously described.

```
#####
# The main application class
#####
class WebRelayMin(object):

    def __init__(self, name):
        self.__name = name

    @cherrypy.expose
    def index(self):
        return "%s - Minimal Web Application" % (self.__name)

    @cherrypy.expose
    def set_relay(self, relay='1', state='off'):
        print("Setting relay %s to state %s." % (relay, state))
        GPIO.set_channel(int(relay), state)
        return 'Relays set!'
```

This is the only code that is called by your web browser. The statements @cherrypy.expose tells CherryPy that these methods will be exposed to the HTTP requests. The method index() is called when you point your browser at the correct URL. In this case, it really does little as our web page just consists of the message returned by index().

The code is slightly simplified as we are yet to discuss the pin map in more detail. The code assumes a straight 1:1 pin mapping.

```
# Entry point
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print("Please provide a configuration file path!")
    else:
```

```
conf_path = sys.argv[1]
if os.path.isfile(conf_path):
    try:
        with open(conf_path) as json_data_file:
            app_conf = json.load(json_data_file)
    except:
        print("Unable to load Json configuration file!")
        sys.exit()

    name = app_conf["name"]
    num_relays = app_conf["num_relays"]
    pin_map = app_conf["pin_map"]
    inverse = app_conf["inverse"]

    # Create web relay instance
    GPIO = webrelay_gpio.GPIOControl(num_relays, pin_map, inverse)

    # Get configuration file
    cherrypy_conf = os.path.join(os.path.dirname(__file__), 'cherrypy_min.conf')
    # Start
    cherrypy.quickstart(WebRelayMin(name), config=cherrypy_conf)
else:
    print("Configuration file not found!")
```

When we write a Python module, we can arrange to run it as a script by including the statement if `__name__ == '__main__'`: When we run this from the command line, Python will automatically start execution at this point.

We need to provide the script with a configuration file path. This is the file I have for a 4 relay module (see /conf/portsw.conf in the download).

```
{"name": "Port Switch", "num_relays": 4, "pin_map": [4,17,27,22], "inverse": false}
```

This file provides the three parameters required by the GPIO module: a name, the pin map and whether the board is active high or low. The first section of code loads this file which is a Json format file but actually looks identical to a Python dictionary. We end up with the variable `app_conf` containing the dictionary, now as a Python dictionary, which means in the next 4 lines we can extract the variables that we need from the structure. Next we create the `webrelay_gpio` instance passing it the three required parameters. The next bit gets the cherrypy configuration file. This is really simple for this

minimal web application:

```
[global]
server.socket_host = "0.0.0.0"
server.socket_port = 8080
server.thread_pool = 10
```

This means that the web server (CherryPy uses a built-in web server which is more than adequate for our needs) should listen on all interfaces and on port 8080. You can change this to listen on a different port if that conflicts with other services.

The final line:

```
cherryypy.quickstart (WebRelayMin (name, config=cherryypy_conf))
```

is an easy way to start the server listening. It specifies to start using the class WebServerMin. We pass this the name so that it can send it as part of its response. It also tells CherryPy which configuration file it should use.

We can start the web server and then connect to it from any machine on the network and drive relays in a slightly laborious way, but then it's intended for automation (ie another application) not a human user. However, it demonstrates how simple the code can be.

Start the minimal web server application. In my case the project files are under /Projects:

```
$ cd /home/pi/Projects/ RpiWebRelay/src/python
$ python3 webrelay_min.py conf/portsw.conf
```

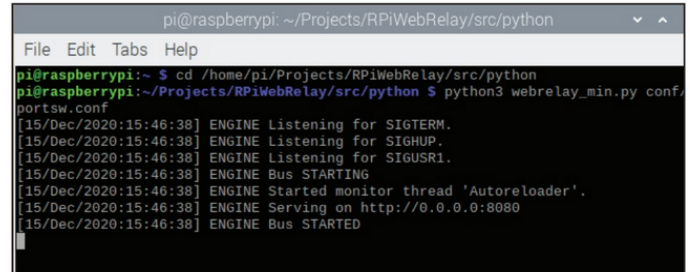
The RPi end of the conversation is shown in **Figure 1**. If we just point the browser at the RPi on port 8080, we get the index page that just returns a string (**Figure 2**). If we then send the query parameters

```
http://<RPi IP address>:8080/set_channel?relay=1;state=on
```

relay 1 will be energised (**Figure 3**).

It is easy to use Python to send the command. This method can be used from within another application to set the relays, e.g. to select the appropriate antenna using the port switch:

```
import urllib.request
def set_web_relay(ip, port, relay, state):
    urllib.request.urlopen('http://%s:%d/set_channel?relay=%d;state=%s' %
        (ip,port, relay, state))
```



```
pi@raspberrypi: ~/Projects/RPiWebRelay/src/python
File Edit Tabs Help
pi@raspberrypi:~ $ cd /home/pi/Projects/RPiWebRelay/src/python
pi@raspberrypi:~/Projects/RPiWebRelay/src/python $ python3 webrelay_min.py conf/
portsw.conf
[15/Dec/2020:15:46:38] ENGINE Listening for SIGTERM.
[15/Dec/2020:15:46:38] ENGINE Listening for SIGHUP.
[15/Dec/2020:15:46:38] ENGINE Listening for SIGUSR1.
[15/Dec/2020:15:46:38] ENGINE Bus STARTING
[15/Dec/2020:15:46:38] ENGINE Started monitor thread 'Autoreloader'.
[15/Dec/2020:15:46:38] ENGINE Serving on http://0.0.0.0:8080
[15/Dec/2020:15:46:38] ENGINE Bus STARTED
```

FIGURE 1: RPi output from minimal RPiRelay server.

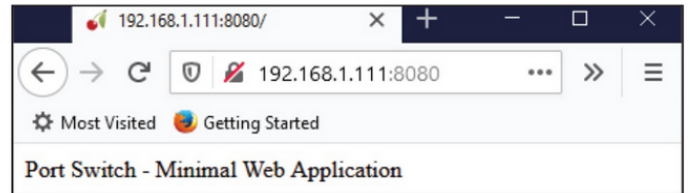


FIGURE 2: Browser output from minimal RPiRelay server.

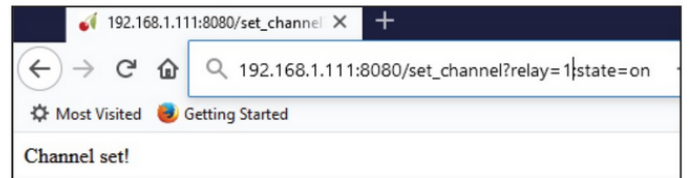


FIGURE 3: Browser output from minimal RPiRelay server when switching a relay.

CONFIGURATION FILES AND PIN MAPS

Often there is a 1 to 1 mapping between an IO pin and a relay when we only need to energise a single relay to make a particular path. This is true of the example project 1 (5V multi-output IP switch) where we can switch on up to eight devices. There is no mutual exclusion required and there are no multi-relay operations.

In the example project 2 (4-way port switch), we don't want more than one path to be active at a time so mutual exclusion needs to be applied. This is the case, for example, when the relays are used for HF antenna switching. In the example project 3 (low-pass filter switching), things get a little more complex still. There are two low-pass filter boards, each of which requires two relay activations to make each path. The pin map, together with the aux_map are designed to deal with these scenarios.

We have been talking about single relay activation so far but, as there can be multiple relay activations for each path, the software uses the term "channel" rather than "relay" and this is reflected in the user interface.

Here is the configuration for the low-pass filter project:

```
{ "name" : "Low Pass Filters", "num_relays": 6, "pin_map": [[27,22,5,6,13,26],  
[18,23,24,25,12,16]], "aux_map": [[4,5,6],[4,17]], "inverse" : true, "exclusive" : true}
```

We have the three parameters previously discussed, **name**, **num_relays** and **pin_map**. In the **pin_map** this is now an array of arrays:

```
[[27,22,5,6,13,26], [18,23,24,25,12,16]]
```

This will cause the software to deal with the relays in pairs. When we energise channel 1, both pins 27 and 18 will be set low/high (depending on the **inverse** setting). In principal, this can be extended to any number of relay sets, limited only by the number of digital pins available. This then accounts for the multiple-relay scenario, but we may also need to deal with extra relays and, in the case of the low-pass filter project, we need to switch between the two boards. This is done by routing the input and output of each board through two more relays.

The **aux_map** parameter tells the software how to manage this:

```
[[4,5,6],[4,17]]
```

The first array is a set of channel identifiers. For the low-pass filter project we have channels 1 to 6 and we should do something for the set 1,2,3 (those not in the array) and something different for the set 4,5,6. The 'something' we need to do is governed by the next array which is another pin map. When we energise channels 1,2 or 3 we should de-energise the relays connected to pins 4 and 17. Likewise when we energise channels 4,5 or 6 we should energise the relays connected to pins 4

and 17. This selects the correct board for the required filter.

Lastly, the **exclusive = true** parameter tells the UI that only one channel can be active at a time. Note that this parameter does not have any effect in the minimal UI as described, so the software must execute a break before make to mimic the exclusive property.

FULL WEB INTERFACE

Whilst the simple interface is useful as an automation method, a full web interface for manual control is provided in the download.

In this case, we start the full web application on the RPI, and we will use the low-pass filter project this time:

```
$ cd /home/pi/Projects/RpiWebRelay/src/python  
$ python3 webrelay.py conf/lpf.conf
```

The index page in our browser now gives us the web form (**Figure 4**). The form will be laid out according to how many channels there are in the configuration. Each channel can be given a description which will be saved in the file **webrelay.model**.

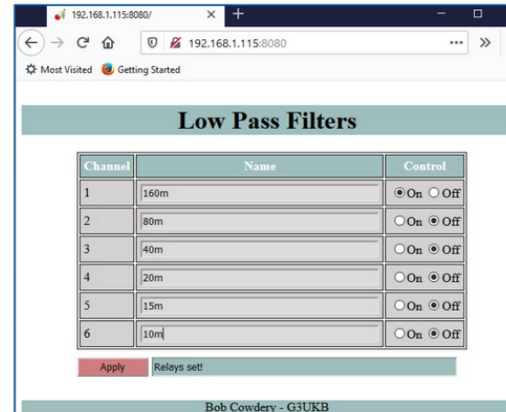


FIGURE 4: Browser output from full RPiRelay server for the low-pass filter project.

WEBSEARCH

- 1: RPIWebRelay: <https://github.com/G3UKB/RPIWebRelay>
- 2: Python: <http://python.org>
- 3: CherryPy: <https://cherrypy.org/>
- 4: Python pip: <https://www.makeuseof.com/tag/install-pip-for-python/>
- 5: 8-relay board: there are many outlets on Amazon and eBay. Search for 8 Channel 5V Relay Module.
- 6: 4-relay board: <https://cpc.farnell.com/sb-components/pirelay/relay-shield-for-raspberry-pi/dp/SC14932?CMP=TREML007-005>

An Automation system for the WSPRLite

The SOTABEAMS WSPRLite and WSPRLite Flexi are useful for their primary purpose of antenna evaluation. WSPR is also used as a propagation guide and it is good fun to see just how far micro-power can get with modest antennas. However, using it in 'fun' mode gets a bit tedious as you have to pre-program the duration of a session, select the correct low-pass filter (LPF) and then press the transmit button at exactly the right second for the start of a cycle. Fortunately, all these manual operations can be accomplished with a Raspberry Pi (RPi), a few other easy-to-source bits of hardware, and a little software know-how.

In this article, I discuss two topics:

1. **How to automate LPF selection.** This uses the RPiRelay project software which is briefly described here in its relation to this project. A fuller explanation of the software with some example projects can be found in the previous article in this edition of RadCom Plus ("Driving relays over a network").
2. **How to run WSPRLite remotely.** Whilst the configuration software available for the device still needs to be used for initial settings, a bespoke application runs on the RPi to set the frequency and start and stop cycles at the correct time.

The software is all written in the popular Python programming language that is both friendly for beginners as well as being powerful enough for seasoned coders.

The WSPRLite might be small, but it is quite a complex piece of equipment. The wire protocol is fairly involved and there is little documentation. However, the configuration program is open source and does provide a working example of how to do things, but it's not an easy read. I have done a minimal implementation that has enough capability to change band and automate transmission cycles.

WARNING: The software will send commands to your WSPRLite. As explained, it uses a subset of the full command repertoire and excludes anything potentially damaging like firmware updates etc. The software also selects a random frequency within the given band, selects the correct LPF filter, and puts the device into transmit. Please do your own risk assessment before using the software!

GETTING THE SOFTWARE

There are two downloads available. The URLs are given in [1] and [2]. (The references are provided at the end of this article.) The RPiWebRelay application is for topic 1, and the WSPRLite for topic 2. You can either go to the link and download a zip file, or you can use git to clone the repository. The latter is the preferred way as it is then easier to keep up to date.

The RPiWebRelay software is a web application. This means that it runs on the RPi controlling the LPF filter selection, so it needs

eventually to reside on your target RPi.

The WSPRLite application is a client/server application. This means there is a server side that runs on the RPi connected to the WSPRLite, and there is a client side that runs on any other device (or even the same device) that is able to run Python. Both the RPiWebRelay and WSPRLite applications can run on the same RPi concurrently, or you can use two separate RPi machines.

In the following, I have given the commands recognised by an RPi, which runs under a Unix type of operating system:

To use git:

```
$ apt-get install git (if not already installed)
$ cd <your directory>
$ git clone <URL>
```

(Here, the triangle braces denote a substitution. Thus, if your directory name is "wsprrlite", then substitute "wsprrlite" for "<your directory>", so that the command line becomes `$ cd wsprrlite.`)

This is not an RPi tutorial and it is assumed you have a model 2, 3 or 4 with the latest *Raspberry OS* installed. The applications are pretty lightweight so a model 2 will do just fine. Almost everything you need will already be installed with the latest *Raspberry Pi OS* except the Python Web framework for the RPiWebRelay software. This is called CherryPy and is easy to install:

```
$ sudo apt-get install python3-pip (if not already installed)
$ pip3 install cherrypy
```

The pip software is available from [3]. (Note that the software uses Python 3, hence you must use pip3 and not pip.)

WSPRLITE LPF AUTOMATION

The WSPRLite Automation project uses the Automation mode available in the RPiWebRelay project, and is built into the WSPRLite Automation software such that the correct filter is selected automatically according to the band in use on the WSPRLite. The associated hardware uses an RPi for control, a 16-channel relay module, and two SOTABEAMS LPF filter modules covering the six main bands. The hardware design can be anything that meets your requirements; as long as it has relay drivers connected to the digital I/O pins of an RPi, the RPiWebRelay software can be configured to run it. The finished hardware unit is shown in **Photo 1**.

As well as providing a simple interface for an application to control relay

Bob Cowdery, G3UKB
bob@bobcowdery.plus.com

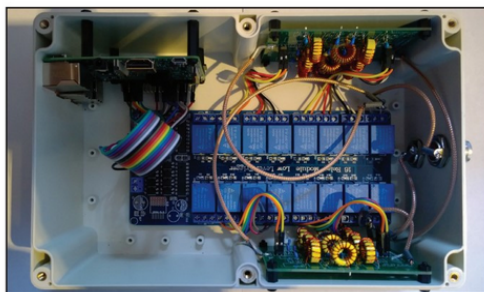


PHOTO 1: Internal view of the LPF selector hardware.

operations, the RPiWebRelay project also has a full web interface for control which should run in any browser on devices from phones to desktops.

WSPRLITE AUTOMATION PROJECT

This project uses the SOTABEAMS WSPRLite Flexi.

I have implemented the code directly from the protocol definition. This is available from the github repository for the WSPRLite Configuration Utility [4]. For those interested, the protocol definition is under techdocs/serial.md.

Run the configuration utility on your usual machine to set up your WSPRLite (see the WSPRLite manual). All configuration options should be set for initial testing: see **Figure 1**. The only parameter that can be set by the Automation software is the frequency. If you have not already tested your WSPRLite, do so using the normal manual process as described in the manual, noting the various stages from the feedback provided by the LED.

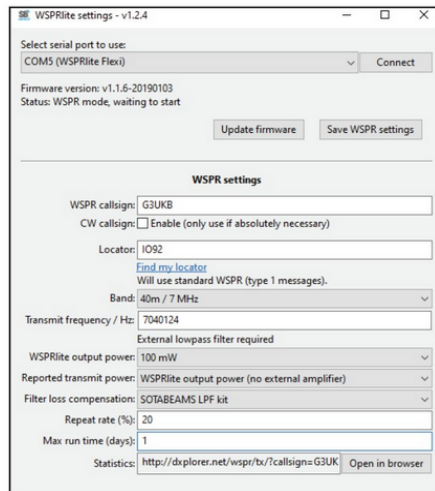


FIGURE 1: Example settings using the WSPRLite configuration utility.

CONFIGURATION

You can make your own configuration file by editing the file common/defs.py in the WSPRLite download. The only fields of interest are the IP addresses and the supported bands. Specify the address of the RPi running the RPiWebRelay software:

```
WEBRELAY_IP = '192.168.1.115'
```

Specify the address of the RPi running the WSPRLite Automation software:

```
SERVER_IP = '192.168.1.115'
```

Specify the bands you want to appear in the drop-down menu. You should only specify the bands for which you have some means of selecting an appropriate filter:

```
BANDS_AVAILABLE = ('160','80','40','20','15','10')
```

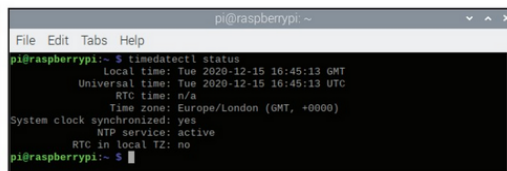


FIGURE 2: RPi output for time synchronisation.

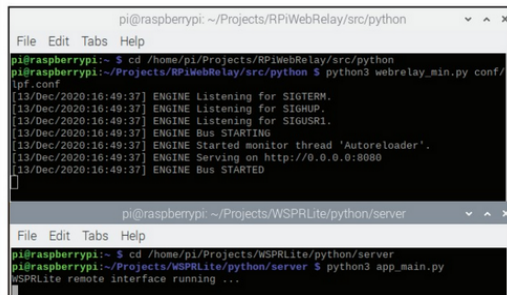


FIGURE 3: RPi startup showing the RPiRelay server and WSPRLite Automation server.

You should also ensure that your RPi has a time service configured and running because accurate time control is essential for WSPR. **Figure 2** shows how to check this. There is plenty of information available on the Web about how to change things if they look wrong, or you suspect that the time isn't properly synchronised.

SERVER (RPI SIDE)

Connect up the system having tested the unit manually. It is assumed that you have either built the hardware for the RPiWebRelay project or have some alternative means of selecting the appropriate LPF. I run both applications on the same RPi. The LPF project is built into the enclosure shown in Photo 1. The WSPRLite Flexi is simply connected to one of its USB ports and its antenna connection goes via the LPF filters.

Figure 3 shows two command windows running on the RPi just after starting up. The upper console shows the starting up of CherryPy for the RPiWebRelay software. The lower one displays the starting up of the WSPRLite Automation server software.

The command windows show the location of the software on my RPi. If you simply clone or extract the zip file into `~/home/Projects`, you will end up with same

```

Command Prompt - python app_main_client.py
Microsoft Windows [Version 10.0.18363.1256]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\User>cd E:\Projects\WSPRLite\trunk\python\client
C:\Users\User>:

E:\Projects\WSPRLite\trunk\python\client>python app_main_client.py
WSPRLite Automation Client running...

```

FIGURE 4: Python output starting the WSPRLite Automation client.

directory tree as I used. I have shown a manual start-up here, but you could make a shell script and launch both applications from a desktop icon. (Note that you must use python3 when starting the applications. You will be greeted with an error if you type “python” and not “python3” by mistake!)

CLIENT SIDE

Having started the RPi applications, the client can be started on any machine on the network that can run Python, including the same or another RPi. Figure 4 shows the client started on my Windows 10 machine. You can run client and server sides in any order and stop or start either side as required. Type Ctrl-C to terminate either application. Note if you Ctrl-C the client the server will be terminated as well.

The client user interface (UI) will initially appear as shown in Figure 5 and is straightforward.

The **status area** shows the callsign, locator and the current transmit frequency. This area is read-only. The data are returned from the WSPRLite, so the server side must be running, else these fields will be blank except for frequency which will show 0.0.

The **control area** has several fields:

Set Freq allows you to set a specific frequency, but will only accept frequencies within the WSPR bands for which you have LPFs. If you try to set an invalid frequency, you will be presented with a message box, as shown in Figure 6.

Band allows you to specify a frequency band. When set, a random frequency within that WSPR band will be selected and set which will then appear in the status area, read from the WSPRLite. If you don't like the frequency, then select **Band** again, or set your own frequency as above.

TX: To start transmission cycles, click the **Start** button. The interface should then move from the IDLE state to the WAIT-START state, shown in Figure 7. In this state, the server(RPi side) is waiting for the start of the next even minute. The clock in the bottom right of the interface is the time on the client device but the timing is being performed on the server side. Now, if both are synchronised to a time source, they should coincide. Once the next even minute begins, the status will move to TX-CYCLING, as shown in Figure 8.

TX cont: Click the **Stop** button (same button, but now with a different label) to stop transmissions.

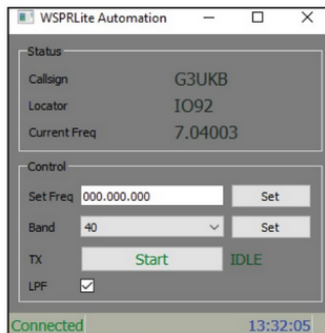


FIGURE 5: Client application in the idle state.

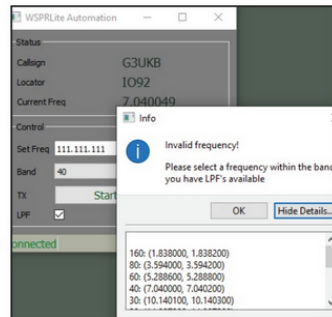


FIGURE 6: Client output when selecting an incorrect frequency.

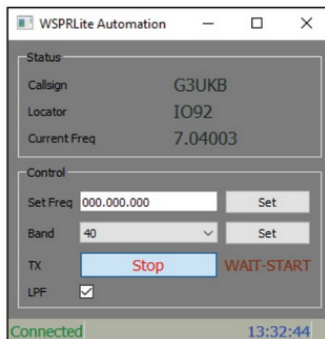


FIGURE 7: Client application waiting for the next start time.

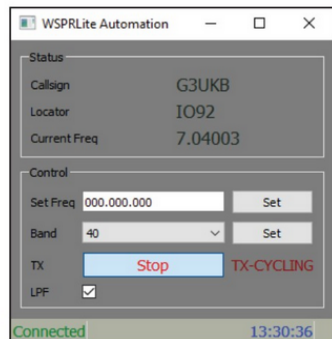


FIGURE 8: Client application running Tx cycles.

The state will move to WAIT_STOP (Figure 9). The machine will wait for the current cycle to finish (just before the next even minute) and the state will then revert to IDLE.

LPF: This is checked by default, meaning that the inbuilt LPF activation will execute. If you have a different arrangement for filters, then uncheck this box as otherwise it will attempt to talk to

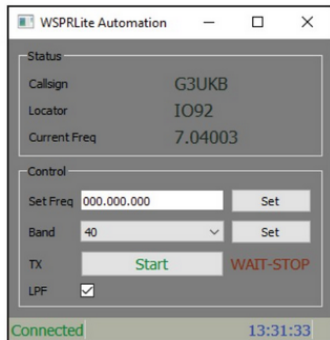


FIGURE 9: Client application waiting for stop time.



PHOTO 2: The connected system in the shack.

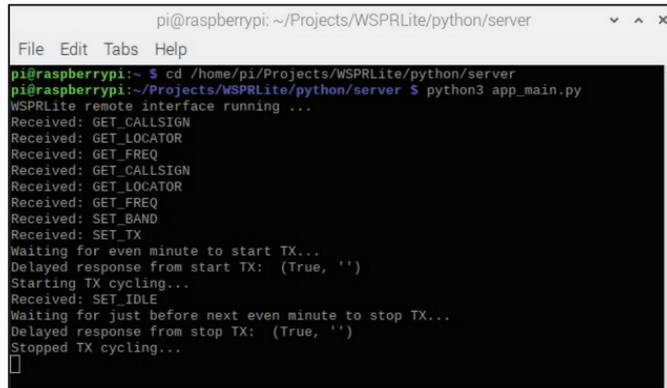


FIGURE 10: Server output on the RPi reflects its current state.

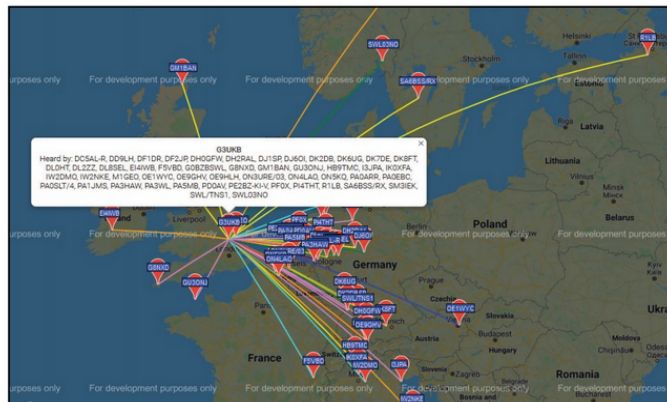


FIGURE 11: WSPRnet showing results of a brief 40m test.

the RpiWebRelay software, timeout, and then display an alert message.

Figure 10 shows how the RPi follows along with messages indicating its state.

DEPLOYMENT

The three programs, i.e. the LPF selection, the client side, and the server side of WSPRLite Automation, can be run on the same RPi for a self-contained system. This could be accessed remotely from some other device by running virtual network computing software (VNC) [5] in order to start execution. At the other end of the scale, it can also run across 3 devices, i.e. the LPF selection on one RPi, the WSPRLite Automation server on another, and the WSPRLite Automation client on, say, a desktop or laptop device. Personally, I am running this at present across 2 devices. An RPi runs the LPF selection and the WSPRLite Automation server, residing in the shack for testing (Photo 2). The WSPRLite client runs on my Windows 10 desktop in the house.

The WSPR map shown in Figure 11 was the result of a quick 10 minute test with a modest end-fed antenna. From my perspective, the intention is to be able to flip this to any antenna on any band with everything running remotely. To this end, the antenna output of the low pass filters is plugged into my Flexi-Antenna switch that was described in *RadCom* Dec 2020 and Jan 2021. The configuration for this is shown in Figure 12.

This project is also related to a WSPR project that was published in *RadCom* May 2018. That project described the use of an RPi for the actual WSPR transmission itself and a FUNcube Dongle Pro for reception. That was in many ways a more complex project, with a scripting engine and automatic tuning and switching of

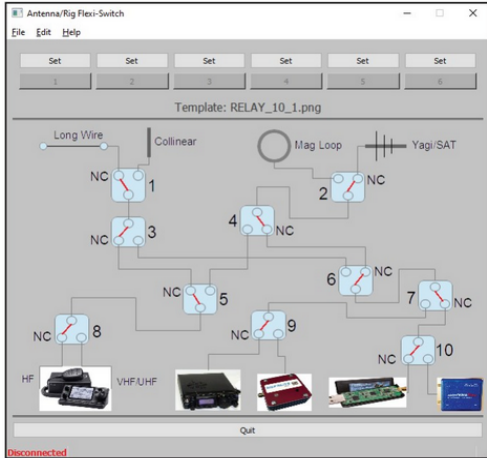


FIGURE 12: Antenna (FlexiSwitch) connections showing the WSPRLite.

antennas. At the time, it was my intention to replace the transmission side with a WSPRLite Flexi. The two projects will probably merge in the future. The hardware was a little "Heath Robinson" and was limited to 3 bands (**Photo 3**).

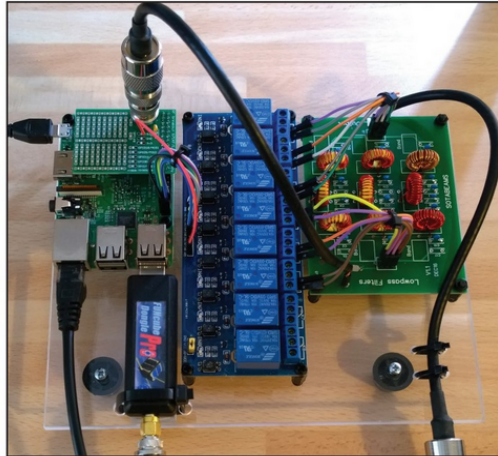


PHOTO 3: Related, previous WSPR project.

WEB SEARCH

- 1: *RPIWebRelay*: <https://github.com/G3UKB/RPIWebRelay>
- 2: *WSPRLite Automation*: <https://github.com/G3UKB/WSPRLite>
- 3: *Python pip*: <https://www.makeuseof.com/tag/install-pip-for-python/>
- 4: *WSPRLite Configuration Utility repository*: <https://github.com/SOTAbeams/WSPRLite-Config>
- 5: *VNC*: <https://www.raspberrypi.org/documentation/remote-access/vnc/>

Sporadic-E – where we are now?

Sporadic-E or Es is a propagation mode where the unexpected often seems to happen and can magically appear and disappear without rhyme or reason. This, I think, is no longer the case since a great deal of progress has been made in our understanding during the last two decades about the behaviour of mid-latitude Es. The combination of professional research plus thousands of amateur radio logs means that we now know enough to be able to propose possible reasons for many of the events, and even to be ready on the bands for the openings to develop. It is a truly exciting mode to operate when a dead VHF band is transformed into a busy 20m-style frenzy of contest activity.

It is a basic rule in life that the better you understand how something works, the more you can gain from it, and this is especially true of Sporadic-E propagation. This is not a simple task because Sporadic-E is a very complex propagation mode and it will take a dedicated read of this article to get the insights that are available to enhance your operating, but it is definitely worth the effort. It is undoubtedly true that much of the amateur knowledge base, built up over many years of operating, is only there because you take the time and effort to report your QSOs to the many different online resources... thank you for the hugely valuable

contribution over the years.

This may be a good time to say to HF operators that you are also able to benefit from this increased knowledge of how Es works. In fact, Es is probably more common on HF than on VHF, but most people won't define it as such. It may be a case of short skip on 20m or it can even make a big difference to local 80m nets, for example. The best starting point is to assume that Es can play a role all the way up to and including 2m, but the main bands we traditionally think of for Sporadic-E events are 10m, 6m, 4m and 2m.

There are many ways of finding out about where and when Es occurs with numerous web sites either showing maps of paths worked or plots of estimated Es probability. These, alongside the standard operating craft skills of monitoring the bands, beacons and clusters, should give you a good picture of what is happening. This article will now cover the current thinking of how Sporadic-E is formed and how to use the many online resources to maximum advantage.

OBSERVATIONAL HISTORY

There is a certain amount of 'folklore' about Sporadic-E within the amateur community, and I suspect most of this will have been

born of a partial fact at some point along the way. It's important to find out which is real research-backed guidance and if there are some limitations as to when, or how, these rules can be applied. This section will examine the main background facts about where and when Sporadic-E occurs, a sort of climatology of Es, if you like. We can then use this observational data to support or negate some of the popular ideas about Es.

When we examine the many years of amateur logs, the data present a fairly well-known picture of how we imagine the Es season to be (see **Figure 1**). There is a distinct seasonal variability, with a summer peak in June within a broad period of activity starting late April on 10m, and gaining momentum with increasing frequency up to 2m before winding down in early September. There is also a strong body of evidence that we quite often see a minor uptick in Es activity in mid winter, during December and January. It is also worth noting that

Jim Bacon G3YLA
g3yla@hotmail.co.uk

the advent of FT8 has revealed a much broader period of activity, and for digital modes there are probably few parts of the year when Es of some form or other is not available. If forced to define 'lean months' then I would suggest February and March plus October and November.

During the day, activity logs show there is also a marked signature for two periods of

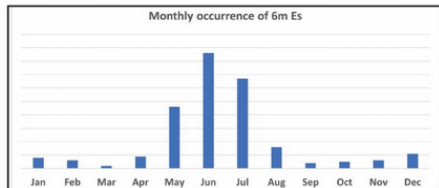


FIGURE 1: Variation of Es by month.

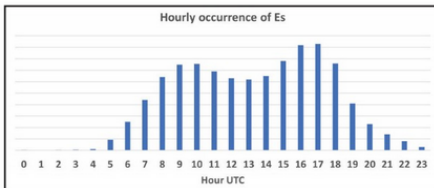


FIGURE 2: Variation of Es by hour.

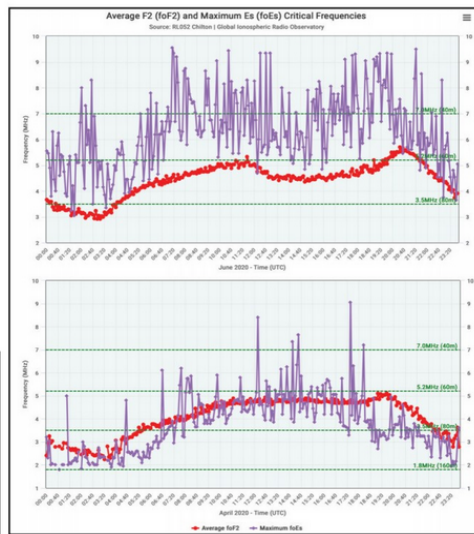


FIGURE 3: Maximum foEs by hour June/April.

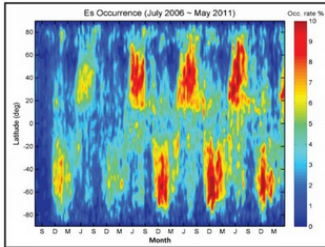


FIGURE 4: Annual Es variability.

stronger activity during the main summer season as shown in **Figure 2**; one in the morning and then late afternoon or early evening.

You can get a sense of what the diurnal variability looks like by viewing the Averages section of the Propquest website [1], which is described in more detail later. A scan through the different months will show the change of pattern of the maximum foEs (the penetration threshold frequency of the ordinary ray at vertical incidence for Es) between the peak season and the quieter months (see **Figure 3**), in addition to

a change of about 2MHz between summer and winter, and probably worth further investigation.

A research project using the COSMIC global positioning satellite network (GPS) and limb sounding with low Earth orbit (LEO) satellites brings some very interesting colour to the longer-term global climatology of Es [2]. It shows broad mid-latitude bands of Es, and a general absence of Es over the magnetic equator. It also shows a marked variation between some years, which appears to be mirrored in the opposite hemisphere, so it's a global influence at play and references suggest that it is related to changes in the meteor input (**Figure 4**).

SPORADIC-E SCHEMATIC

The main problem in trying to understand Es is that it soon becomes apparent that it is a system of many possible components – see **Figure 5**. A fundamental point is that it is very difficult to develop operating rules if you try to base your understanding upon just one favourite factor such as meteors or, say, thunderstorms. The importance of these different factors is not equal, but we can now begin a closer look at where the principal indicators may be found,

and importantly where the radio amateur can find information about these determinant components. Note that some of these components may be positive influences, and others negative influences, on Es.

Several years ago, it became apparent that Sporadic-E is a very complex business, and focussing on one feature was never going to allow a working understanding to aid operating during the Es season. I developed this schematic (**Figure 5**) to provide some order for me to try to deal with components separately, whilst still retaining the 'big picture' concept. It is helpful to consider three main regions of interest:

E region: complex interactions of the elements of the wind shear theory in the E region;

Bridging the gap: the large vertical interval between weather and the E region; and

Weather: the source of atmospheric gravity waves (AGW) generated by the weather.

I suggest that, if you keep this schematic in mind when working Sporadic-E, it may be less of a mystery. As we will find, many elements are now accepted science and, although I am approaching this from the amateur radio

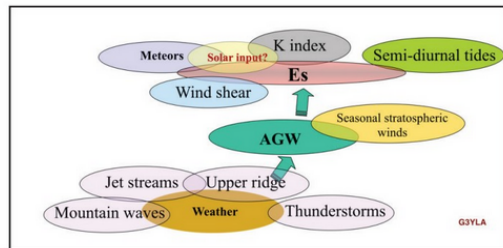


FIGURE 5: Sporadic-E schematic.

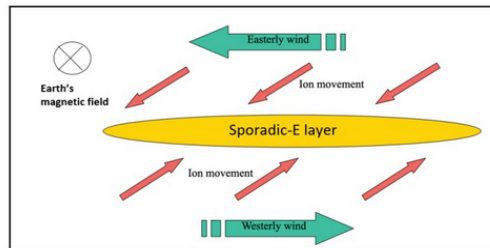


FIGURE 6: Wind shear theory of Sporadic-E.

perspective, this is also a hot topic within the research community with new papers being published regularly. It is by no means a completed project, and there is still vigorous debate on several aspects of Es, but we radio amateurs have the big advantage of regular new data streams each Sporadic-E season, highlighting new paths or rejecting possible links.

We will now start our examination of the various ingredients in the E region and the key to the whole process, the wind shear theory of Sporadic-E. Then work our way down to the Earth's surface to see if any links exist to explain the sporadic geographical nature of Es.

E REGION: THE WIND-SHEAR THEORY OF SPORADIC-E

The currently-held view goes back to work by Whitehead and Dungey during the 1950s and 1960s, and is known as the wind-shear theory of Sporadic-E. There has been much follow up work over the years [3]. The essential ingredient is that winds, or more correctly, **differences** in winds, produce a wind shear, defined as a change of speed or direction. There can be wind shears in both the horizontal direction and the vertical direction. For Sporadic-E formation, we are more interested in wind shear in the vertical direction, specifically changes in horizontal wind speed or direction with altitude. A book chapter tutorial by Haldoupis [4] gives a very good description of the wind shear theory. In an ideal case, vertical wind shear of neutral horizontal winds (an easterly wind above a westerly wind), can move charged particles vertically by causing them to converge into denser layers, which are of course the Sporadic-E patches we use on the bands – see **Figure 6**.

Note that in many ionospheric texts, winds

are referred to by the direction they are blowing towards, rather than in meteorology and public forecasts which use the direction they are coming from. A westward wind in ionospheric physics is the same as an easterly wind in a weather forecast!

It is important to realise that the reason the wind shear can operate in this way depends upon the fact that the Earth has a magnetic field. This obviously runs from magnetic pole to magnetic pole and exhibits a dip angle ranging from horizontal at the magnetic equator to vertical at the magnetic poles. In mid latitudes, the angle of dip is typically around 55-65 degrees. Zonal (E-W) neutral winds in the E region, where the meteors burn up will, in effect, be moving the ionised meteor debris across the Earth's magnetic field and, with the right vertical wind shear, the ionisation will be forced to converge vertically (or more correctly experience a geomagnetic Lorentz force [3], [4]) to form nodes of maximum and minimum ionisation density spaced according to the vertical wavelength of the neutral wind regime.

The long lifetime of the metallic meteor ionisation gives ample opportunity for the wind shear to converge these ions into a denser layer. The dissociated electrons follow the positive metallic ions to maintain regional charge neutrality, and the resulting enhanced electron density gives us the Sporadic-E patches much used by radio amateurs. The wind shear process does not operate at the same rate across all heights. The zonal (E-W) winds are the important ones for Sporadic-E and in reference [4] we are shown a very interesting graph of the vertical convergence time over a distance of 1km. In fact the Zonal winds at an altitude of 110km take 10mins to move the ions 1km, whereas meridional winds (N-S) take

100 minutes for the same result. The quickest formation times occur at around 125km, which is nearer the upper height of Es patches at the start of an opening, giving maximum range. Equally, it is also the height at which Es is most quickly destroyed by the negative effect of a divergent vertical wind pattern (westerly above easterly winds). This means that the maximum-range higher Es patches are likely to be more fickle than those lower down nearer 100km. An important trait of Sporadic-E layers is that they descend with time, which we will discuss later and, in light of the above, it probably suggests that the longer an Es layer lasts, the longer it is likely to last, given its gradual descent. Once a Sporadic-E layer descends below about 95km, chemical processes change and the ionisation is no longer sufficient to maintain the Es.

We will examine the cause of the wind shear in two later sections on semidiurnal tides and atmospheric gravity waves.

E REGION: METEORS

Meteor input is widely regarded as the 'fuel' for Es. As meteors burn up in the lower thermosphere and mesosphere, the ionisation provides long-lived metallic ions of magnesium and iron among other elements. Over the years, the general focus in the amateur radio community has largely been driven by the dates of major meteor showers. In many ways, this is a good default position regarding Sporadic-E, since we know that meteors are the sources of the ionisation, but it may not be quite the right thing to monitor.

The accepted view, from various rocket experiments which have been launched into Sporadic-E clouds, is that they are composed of ablated debris (mainly iron and magnesium) from meteoroids as they burn

up in the Mesosphere and Lower Thermosphere, known as the MLT region, at around 70-120km. The monatomic metallic ions have lifetimes of the order of several days above 100km and can even be transported to over 400km height. A series of chemical reactions can maintain the supply of monatomic Fe+ ions above 100km, despite dissociative electron recombination rates [5].

Although a full list of meteor showers is quite long, some have very low zenith hourly rates and some have low entry velocities, which suggest lower metallic ion potential. As an experiment, I suggest a crude meteor shower 'quality factor' (MSQF) to see which showers are likely to be the most significant in terms of potential supply of long-lived metallic ions. This expression includes the maximum zenithal hourly rate (ZHR):

$$\text{MSQF} = \text{velocity} \times \text{ZHR} \times \text{brightness},$$

where brightness is equal to 0.5 for weak meteors, 1.0 for medium meteors, and 1.5 for bright meteors.

Out of 37 showers checked, I found four which are significantly above the rest:

- 3rd January Quadrantids, MSQF=6765;
- 5th May Eta Aquarids, MQSF=4950;
- 12th August Perseids, MQSF=8850; and
- 14th December Geminids, MQSF=5250.

The remaining showers have quality factors below 2475.

Note that if the other elements of the

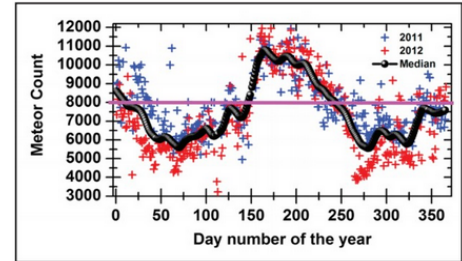


FIGURE 7: Daily meteor count 2011/2012.

schematic are not favourable, the ionisation may not be converged into trackable Es and a meteor shower will not be detected in the Sporadic-E response. This all highlights the risk of making too close a link to a given parameter without being aware of what the other defining factors are doing.

At this point we should probably loosen our fixation with meteor shower dates. Research shows that the mass of metallic ions entering the Earth's atmosphere from the random background input, probably due to debris from the asteroid belt and ancient comets, is far greater than the input from narrowly-defined meteor showers as the Earth's orbit passes through the trails of comets [6]. The background meteor input also better describes the Sporadic-E season. Since the lifetime of the ionisation produced when meteors ablate or burn up is relatively long, of the order of several days, any relationship to shower dates will be blurred, at best. Something to remember for the later section on semidiurnal tides is that the maximum meteor ionisation input tends to be just before local dawn when that part of the Earth is facing forwards in its orbit around the Sun and entry velocities are higher.

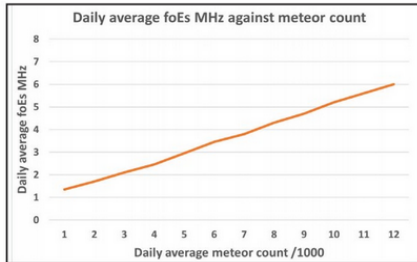


FIGURE 8: Average daily mean foEs plotted against average daily meteor count.

In a research paper by Selvaraj [7], a graph is presented (reproduced in **Figure 7**) which shows background daily meteor input at a low latitude site, although I'm using this as a relative daily indicator for mid latitude regions. Firstly, notice there is a striking variation from one month to the next and even more extreme is the daily variation between successive days or the same day on successive years. Although the graph data only show two years of measurements, the contrasts are very significant, notwithstanding the blurring due to the long-lived nature of the ionisation. This should caution against assuming a date repeat factor for meteor input from year to year, other than in very general terms.

However, there is something we can do to gain operating advantage from just these limited data sets. If we nominate some arbitrary value of daily background meteor count as being significant for Es on the amateur bands, say 8000 per day, then it's possible to find dates when this limit is reached. This turns out to fit fairly well with our current understanding of Es occurrence on the amateur bands. An early January window from 1st-15th, and the main summer season from 24th May to

6th September, are above this arbitrary threshold, based upon the average of just the two years of data. Obviously, there are many limitations of such a small sample, but it is a useful guide and reaffirms our subjective ideas about the Es seasonal variations.

A study by Haldoupis [6] shows a relationship between background meteor input and foEs in a mid-latitude setting. A schematic graph based upon the paper is shown in **Figure 8**; note that the paper only plots up to 8000 counts per

day, whereas for radio Es we see significant results between 8000 and 12000 counts per day (Figure 7). This highlights the problem of averaging a daily foEs and meteor count over 6 years, useful for climatology, but not capturing the extremes of a specific Sporadic-E event. By extrapolation, the relationship might suggest average daily foEs of about 6MHz for average daily meteor count of 12000 per day.

E REGION: SEMIDIURNAL TIDES

We now need to look at the winds that form a critical part of the wind shear theory, and we can start by examining the normal tidal winds which exist at these heights. As each part of the Earth faces the Sun, the atmosphere above it is heated. This produces differential pressures at the height of the E region, and these in turn produce a wind flow. Because this wind flow varies as the Earth rotates, it is referred to as a tidal wind (although not having anything to do with the Moon). The semi-diurnal tide appears the more dominant at mid latitudes and gives a peak twice per day, although the diurnal tide also features. In reality, the varying

winds in the MLT region where Sporadic-E is found are a combination of many different periods of tidal effects. Satellite data show there are contributions from 24hr, 8hr and 6hr components, in addition to the 12hr semi-diurnal tide.

The above are known as migrating tides since they follow the Sun, but there are also interactions with non-migrating tides or fixed effects due to differential heating over oceans and land masses [8]. These longitudinal variations in the semi-diurnal tidal winds may go some way to explaining why there can be variations between activity in, say, Europe and the USA. Satellite and ground-based radar studies have also shown substantial variations in the magnitude of the semi-diurnal tides from day to day, another reason why Es conditions may vary despite the other main ingredients such as weather triggers being similar to the previous day.

As we discovered in the section on wind shear, the zonal E-W winds are important. In reality, we are hostage to the variable nature of the semi-diurnal tides (their speeds can be as low as 5ms⁻¹ and as high as 80ms⁻¹) plus any possible interactions with other tidal periods. Other multi-day signatures can be found in these MLT winds and are known as planetary waves. These may influence the chance of repeat Es on subsequent days as discussed by Chris Barnes GW4BZD [9]. One of the signatures of such tidal influences is that, looking vertically upwards, they produce a rotation of wind direction; imagine that it looks a bit like circular polarisation (**Figure 9**). This can produce, at half vertical wavelength separation, opposing easterly and westerly winds as required for the wind shear theory of Sporadic-E (Figure 6), with a convergent zone of ionisation in between

them. A further feature of this class of tidal wave activity is that the nodes move downwards with time, and so we should expect the Es layers to do the same.

However, there is a consistent signal in the climatology [2] of a twin peak in Es activity during the day, particularly in the summer. There are occasions when this is not seen, and we just have the afternoon/evening opening on the edges of the main summer season. However, perhaps contrary to results in [2], single openings around the middle of the day have also cropped up outside the main season in radio amateur activity. It may be that a more-direct link to solar irradiance may sometimes play a role. You can see this effect in the averaged data of the maximum foEs, shown in Propquest [1]; try comparing December with June, for example. This will be an interesting area of further study when a longer data series

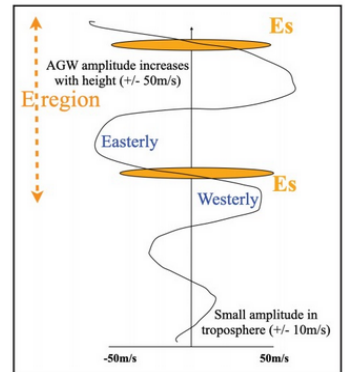


FIGURE 9: Varying wind direction with height of gravity waves.

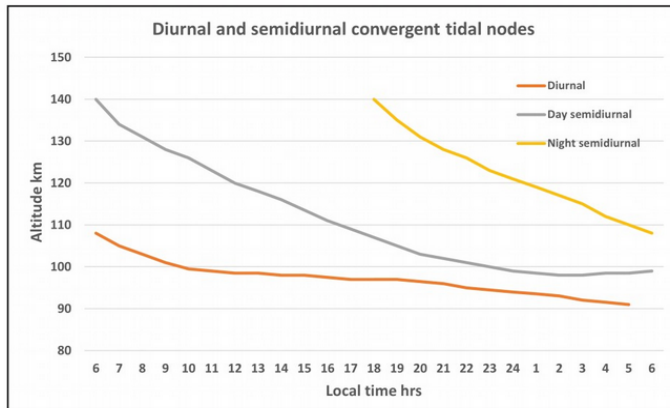


FIGURE 10: Schematic of semi-diurnal and diurnal tidal Es layers.

becomes available. The challenge of the wind shear theory of Sporadic-E is to see if this double peak can be explained by the descending nodes of the semidiurnal tidal winds. Results from the Arecibo incoherent scatter radar [4] show clear evidence of the expected descending layers of Es, compatible with the semidiurnal tides, along with a layer formed by the diurnal tide shown in the schematic of Figure 10, which is based upon radar plots from [4].

The schematic shows a 24 hour period from 6am to 6am local time. The lower trace is the slowly descending diurnal tidal node and appears remarkably persistent on the radar measurements [4]. It is not entirely clear in an amateur radio sense whether the diurnal tide contributes much to our openings in view of the double peak in Es activity, but who knows what the revelations of FT8 will bring.

The middle trace is the day semidiurnal tide which shows a descending layer of ionisation trapped in a convergent wind shear node. The start time just after 6am shows a fairly rapid descent initially, but then slows down. It has the advantage of being able to sweep up the recent pre-dawn input of 'meteor fuel' and by the time it gets down between 110km and 100km should be able to provide the best chance of enough ionisation for Sporadic-E contacts and ties in very well with the traditional late afternoon or early evening opening window.

The upper trace is the night semidiurnal tidal node, which behaves in a similar way to the daytime progression and reaches the ideal altitude of 110km to 100km in the early morning, possibly merging with the diurnal tide. This provides a window of opportunity from about 6am to 10am and again fits well with

the morning Sporadic-E peak. The night-time node has thin pickings following on behind the day-time node, which has probably scooped up most of the pre-dawn meteor input from that morning, but can get a boost at the end of the following night with the next 'delivery' albeit from a reduced altitude range by that time.

To my mind, it seems that the two semidiurnal tidal nodes can explain the double peak in Sporadic-E activity and even offer a reason why the morning event is usually weaker than the afternoon or early evening events. Closer examination of actual openings in relation to heights, h'Es, and ionisation, foEs, from the ionosonde network should provide some clarity here.

The stronger ionisation in this descending convergent wind shear node is therefore lower down, say 110km-100km, and fits well with the late-afternoon, early-evening window of Sporadic-E activity. At this stage, we need to remind ourselves that ionisation happens quicker at 135km and slower at 100km.

Eventually the semi-diurnal tidal nodes, which continue to descend at the same pace, 'runs ahead' of the ability of the ionisation to converge the metallic ions and so the layers stop intensifying, and may even thin, as a new divergent node arrives, to mark the end of the Es window. However, we know from experience that these timings are approximate with many variables which could either prolong the opening towards midnight or curtail the event early by changing other parameters.

As we have discovered above, there are multiple tidal-wave periods producing the resultant variable neutral (uncharged) tidal winds in the E region, but there are also longer period planetary waves, which may influence the strength of Es over timescales of a few days [9].

The end result of all this wave interaction will usually mean that the dominant period, which in the case of Sporadic-E is the semidiurnal tide, followed by the diurnal tide, presents itself. In the schematic of Figure 10, based on the Arecibo study [3], the diurnal component is seen as a very gently-descending node from about 110km to 90km. This can be present for much of the 24 hours, but perhaps isn't active in an amateur radio sense; or is this where FT8 comes into play? We also know that the convergence rates are very low at lower heights, but so are the dissipation rates, unless the event gets below 90km where we start to see the effects of molecular recombination because of the increasing air density.

Several questions remain regarding the full role played by these lower thermosphere and mesosphere winds in the formation of Sporadic-E. Firstly, the semidiurnal tide operates on an hemispheric scale (the side facing the sun) and as far as I'm aware does not necessarily contain the sub-regional variations required to explain the location-specific properties of amateur radio Sporadic-E. Tides mentioned above vary with latitude and season, but I don't feel they are the only drivers of the summertime maximum for Es.

One outstanding feature to explain therefore is the geographically-focussed nature of Es, so it cannot simply be the tidal neutral winds that are responsible for amateur radio QSOs, since that would imply a widespread event. For the geographical influence, it appears that atmospheric gravity waves (AGW) play a major part, since these are strongly correlated with the location of Es and originate in the tropospheric weather events. It is therefore quite possible that an interaction of the semidiurnal and diurnal tidal winds give the temporal characteristic of

Es, and AGW determine where Es happens geographically. As if this isn't complicated enough, it turns out that there are other planetary waves with periods of a few days, which can also influence the tidal effects.

We will look at AGW in more detail later, but I would also like to spend a few lines describing a methodology for visualising how this complex interaction can be followed, and the way it provides a geographically-selective 'illumination' of Es regions of activity (**Figure 11**).

This can be thought of as an illuminated tuning dial pointer from those domestic wireless sets where a beam of light moves across the dial. Think of the semidiurnal tide as that tuning marker moving from east to west as the Earth rotates. This means that the most suitable conditions for Es also move west during an opening, at the rate of rotation of the Earth;

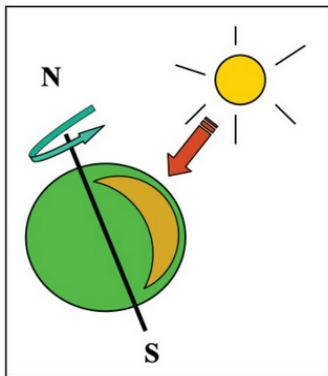


FIGURE 11: Solar-driven Es operating window migration.

this equates to approximately 40 minutes for each 10 degrees of longitude. Thus the ideal conditions (the **wind of opportunity**) for Es over the Balkans will take about 1 hour and 10 minutes to migrate west to Portugal. It is not a single thin line of activity, but better thought of as a broad zone, say plus or minus a couple of hours in width. If the ionisation levels are high enough, there may not be a gap between the two operating windows and, if other parameters become unhelpful, then you may not even get a second window.

What this apparent behaviour is **not** saying is that the Es moves from Bulgaria to Portugal. It merely describes the movement of favourable conditions. It largely explains why the openings often appear to start over Eastern Europe and thence perhaps to Italy before ending up with paths to Iberia, provided that there are suitable weather triggers to produce the AGW component. After a morning event, this whole process may well repeat in the afternoon semidiurnal tidal window, but be aware that the weather triggers may have changed intensity or location in the meantime.

E REGION: K INDEX

The Earth's magnetic field is instrumental in the wind shear theory of Sporadic-E. It is the fundamental reason why ionisation can move vertically when displaced by the neutral winds of the semidiurnal tides and atmospheric gravity waves. A few basics are worth noting for a better understanding of the distribution of Es. The main fact is that the magnetic field varies in its dip angle from zero at the magnetic equator, to 90 degrees at the poles. The main zone of Sporadic-E in the mid-latitudes geographically is through the middle range of dip angles, whereas equatorial Sporadic-E is formed by an entirely

different mechanism. In the polar regions the very steep dip angle can be a zone where auroral Es occurs, but the mechanism is different with the ionisation provided by auroral input rather than by the wind shear mechanism.

If we accept wind shear as the mechanism for making Es, then if the Earth's magnetic field is stable, we should expect good results. However, a wildly-varying horizontal component of the Earth's magnetic field may not give a consistent convergence in the wind shear node. In practice low Kp indices, say 0 to 2, are usually good omens for Es. (The K index is a measure of the disturbance of the Earth's magnetic field and the p suffix denotes a planetary value, but some observatories do record a regional value of the K index.) If a low value increases to 3 or higher, the Es may dissipate or not even form since the convergence rate has been reduced. Kp values of 3 or 4 are marginal, but may still allow Es with the right solar input, while Kp of 5 and above, and especially 7 to 9, may flag the possibility of auroral Es in northern latitudes, but no mid-latitude Es.

E REGION: SOLAR EFFECTS

It is also believed that the solar irradiance, primarily extreme ultra violet (EUV) and X-rays, plays a part too, firstly through the obvious one of its involvement with photo ionisation of the metallic atoms from the background meteor input. This is clearly going to favour strongly a summer maximum and will probably not be hugely different from day to day, but there are times when the solar input can change dramatically.

This is a relatively new item on the schematic and has been added to account for observations of quirky openings over the last few years, which tend to crop up at times of solar flares

and coronal mass ejections (CME) with higher than expected Kp index. This goes against the traditional view of the right conditions for Es. This is a still-evolving part of the story, since there are so few examples to study in the amateur domain, but it seems to be a worthwhile additional area to look at more closely, especially as we head into the new solar cycle 25 and perhaps an occasional enhanced supply of solar-driven ionisation after a solar flare.

BRIDGING THE GAP: ATMOSPHERIC GRAVITY WAVES

The next ingredient we need to consider is whether tidal winds or, more correctly, the wind shears they produce, are the sole influences on Es that we use on the amateur bands. It is hard to see why Es appears to be so geographically focussed if it is just the tidal wind shear that is significant, since it presumably operates on a much larger scale.

There is good evidence in the various MLT radar studies ([2], [3], [6]) of additional wave activity interacting with the tidal components. These are upward-propagating atmospheric gravity waves produced by weather systems in the troposphere. The actual weather features will be covered in more detail in a later section, but the main point is that they are often localised to specific regions and are associated with weather events, and they also tend to move as the weather systems move.

So let us start with a closer look at gravity waves, which in this context are **not** the waves in the gravitational field generated by colliding galaxies! These are more correctly atmospheric gravity waves that is common parlance in meteorological circles and relates to a wave in a fluid (including air) where the displacement

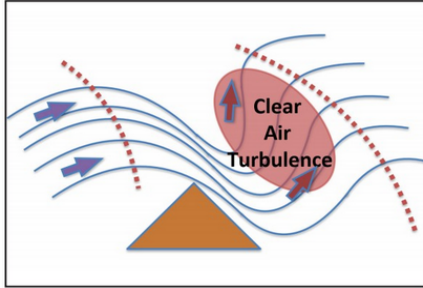


FIGURE 12: Atmospheric Gravity Waves.

is restored by the force of gravity. The right conditions can lead to a train of undulations, say on the surface of the sea, or in the atmosphere as air is displaced when it flows over a mountain. In the discussions about Sporadic-E, we are always talking about atmospheric gravity waves.

The main feature of these AGW is that, as the energy propagates upwards, the amplitude of the wave disturbance grows bigger. The result of this is that we can have a 10ms^{-1} displacement amplified to around 50ms^{-1} by the time it reaches the E region in the lower troposphere. This wave doesn't act in isolation, but modulates the other neutral winds (mainly the 12-hr and 24-hr tides). The likelihood is that the longer timescale descending semidiurnal convergent node of ionisation receives a shorter wavelength disturbance to reinforce the ionisation rate locally. We will see in the discussion of amateur band Sporadic-E openings that the vast majority of events correlate very well with weather features in the troposphere.

One of the features that makes AGW important for Es is that, once a wave train is

initiated, it can continue to produce the undulating wave train for as long as the weather feature is operating. So a jet stream which remains over a mountain range can be an effective source of AGW for long periods, and certainly long enough for the energy to reach up to the E region. The time taken for an initial disturbance to reach the E region is believed to be in the range 20 minutes to 2 hours. We have already heard that the amplitude of AGW grows with height, but complications can arise if they grow

too much and break and shed energy before reaching the E region.

Another interesting feature of AGW is that they tilt into the mean flow and therefore, looking in the vertical direction, you can go from a max to a min and back again at a spacing that is the vertical wavelength, which at E region heights may be around 10-15km (**Figure 12**). This is seen in wind measurements of AGW, where a strong westerly wind can cycle through to zero and then to a strong easterly wind as you move up through half a vertical wavelength. You will remember from our discussion of the wind shear theory that there can be a convergence of ionisation at the crossover point, which descends with time.

It may be that this weather-triggered AGW response, combined with the tidal flow, determines the upper frequency band of Es events, by giving the highest foEs. The recent arrival of the FT8 digital mode has added a whole new population of weaker Es openings, and it would be interesting to explore further to see if they are initiated by just the semidiurnal tidal convergence alone.

BRIDGING THE GAP: STRATOSPHERIC WINDS

There is a very marked difference in the stratospheric wind regime in the summer months compared to the winter months, and thus also between the northern and southern hemispheres. Once again this is another part of the atmosphere where data is hard to come by, but in this instance we are in luck. The mean zonal (W-E) winds in the stratosphere can be followed on the Tokyo Climate Center website **[10]** that gives a regular updated plot for each hemisphere. I follow the 30hPa zonal flow where the colour coding used to differentiate between a westerly flow (yellows) and an easterly (blues) makes it very obvious when the seasonal switch-over occurs for the northern hemisphere mid latitude band. (One hectopascal (hPa) is 100Pa and is a unit of pressure which is the same as one millibar.) We need to look for an end of the strong winter westerly (blowing from the west) winds and the establishment of a weak summer easterly.

This 30hPa zonal mean flow tends to work very well as a proxy for the Es season, although it may not actually be the controlling feature. For example, the change over of the stratospheric wind regime is probably coincident with the increased seasonal heating which also brings a strengthened response in the semi-diurnal tidal winds (the real influential component). So it may be a good visual guide, but not necessarily the direct reason for the improved Es. However, it is also possible that the zonal flow can act as a filter for gravity-wave propagation

from the weather systems below and thus act as a 'switch' for the Es season, but if so this makes the year-round weaker (FT8) events more difficult to understand. A cautionary note is that this parameter is a global zonal mean and thus regional stratospheric winds may not be well represented.

As I write this in early January 2021, we have just seen a good example of a sudden stratospheric warming event (SSW), where temperatures in the stratosphere can rise by many tens of degrees Celsius over 48 hours, in this case about 32C rise. Why mention this? Well it has also been observed to be roughly at the same time as 'out of season' winter Es events. Is this because it reverses the zonal wind pattern (**Figure 13**), to look briefly like the

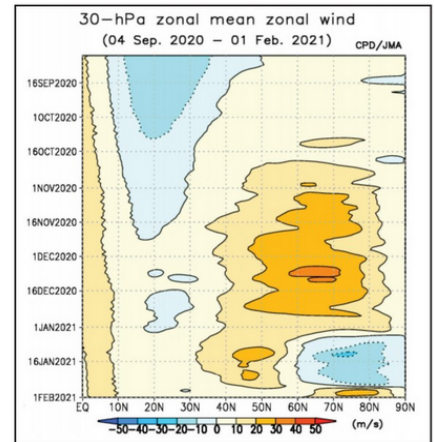


FIGURE 13: Zonal mean winds at 30hPa.

normal summer Es scenario? Possibly, since this was a particularly-strong wind reversal, but it also coincided with the time of the Quadrants meteor shower. This is typical of many of the elements associated with Es: it can be very difficult to isolated separate contributions and it could, of course, be a mixture of them, so there remains work to be done.

I make no apology for spending a lot of time on atmospheric gravity waves and the domain they propagate through because they are the fundamental link between the weather and Sporadic-E. They operate in an area where measurements are very difficult, so in the amateur community we are not blessed with real-time information, but have to rely upon research studies, which may not be fully representative of conditions on the day you find Sporadic-E. Nonetheless, any new research will often add to this background knowledge base and help advance our understanding and predictability of suitable conditions on the amateur bands.

WEATHER

Weather provides a probable source of many of the atmospheric gravity waves which determine the geographical position of Es, and has long been associated with the great Es mystery as a possible ingredient in the mix. The challenge is to find out which items in the weather category are responsible. This can be a win for radio amateurs since weather is reasonably well forecast by the modern mathematical models, so here is a way to get ahead in terms of the locations where Es might form. There are many papers from professional researchers which are worth reading on this subject, but most are rather general in identifying the weather disturbances of interest, since few

examples occur near Mesosphere Stratosphere Thermosphere (MST) radars.

Using the many excellent cluster resources and map displays allows amateurs to make direct associations with real-time Es events, and the locations of possible weather sources in the troposphere. Over the many years of reviewing such events, it is apparent that there are a few particular weather features we can follow by the hour, which give the greatest chance of locating a potential Es development. I will now detail the main players in this very full field.

WEATHER: JET STREAMS

If I had to select a reliable source of AGW in the context of Sporadic-E, I would have to say that jet streams play the most significant part in the archive of openings when cross-checked against the location of these upper winds on weather charts. Fortunately, modern meteorology is very good at forecasting jet streams. Upper air charts for, say 300hPa (**Figure 14**), will be a good place to look for indicators as to where Es might form on the right day. Keep focussed on the fact that this is only one piece of the jigsaw, since it may be that other factors have a negative impact.

A jet stream is a core of strong winds in the upper atmosphere around 10-12km above the ground. They are a representation of the temperature difference across the more familiar weather fronts you see on TV weather maps and the more active a weather front, the stronger the jet stream. Seasonally, these are stronger in the winter at up to 200 knots, while in the summer months may be only in the range 60-80 knots. The irony of this is that, in the Sporadic-E summer season, these potential triggers of atmospheric gravity waves are weaker than in

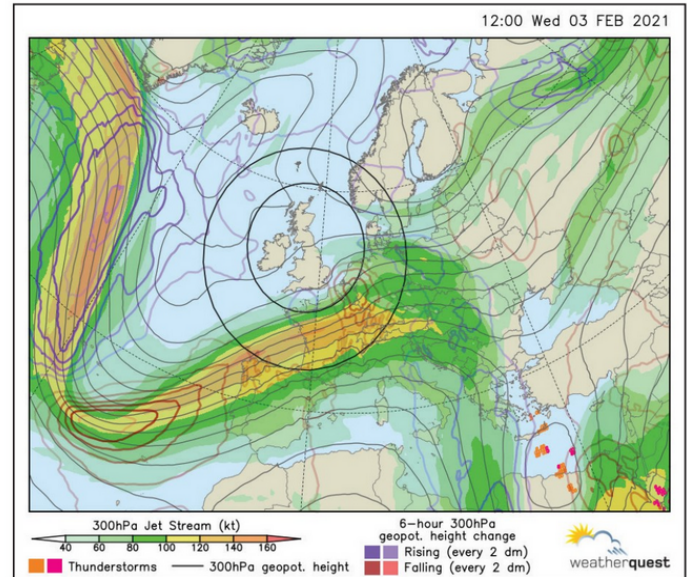


FIGURE 14: 300hPa upper air chart.

the winter 'close-season', but still appear to be sufficient as sources of AGW and certainly would be in any out-of-season winter events, given other factors being favourable.

There is some basic meteorology which will help to make sense of how the contributions from jet streams might vary over time. Jet streams are parts of major weather systems, so are usually well forecast by the models, and you should be able to see from the upper air forecast charts [1] how the jet stream might move during

the day. Note that this is nothing to do with how fast the wind flow is within the jet stream, but all about how the overall pattern translates across the chart. This means that, whilst the position of the jet stream could place Es within range for a morning opening, the pattern could have moved to put any Es out of reach in the typical afternoon/evening opening, or maybe favour a different part of the UK for the home end of the path. Incidentally, it is not necessarily the case that a weather system remains active, so a jet

stream can not only move, but will often change intensity (forecast models are good at predicting this too), making Es more or less likely as a result.

WEATHER: THUNDERSTORMS

It has been often mentioned in Sporadic-E folklore that thunderstorms are associated with Sporadic-E. However, the reality is that these do not account for many of the traditional openings reported on the many cluster sites. Thunderstorms, and the dynamics of their cumulonimbus clouds, are without doubt one of the more dramatic manifestations of weather and do indeed generate strong turbulence, so why do they not feature more strongly in the results?

They do appear to be more effective when moving very slowly, so it could be a timing thing for the AGW to propagate up to the E region. An inconvenience is that most large thunderstorms move through a region fairly quickly, say at

10-20ms⁻¹, and could not generate a sustained AGW train at a particular location. The few that have appeared to be co-located with Es events are usually slowly-moving severe storms, typical of high summer, and more common over the continental landmasses than in the UK. Other disturbances from thunderstorms include acoustic waves and infrasound.

It's also worth reminding ourselves that the weather triggers for launching AGW should be around the mid-point of the path to the DX station for a single hop, and you should not get too fixated by what is happening where you are. These large cumulonimbus clouds are usually driven by daytime heating, and the weather models are pretty good at picking up where and on which day they will develop, although not in an individual cloud sense, but the general area and time of activity. There is one other point worth making: thunderstorms often form ahead of an upper trough pattern where jet stream

winds may also be present, and they are far more likely to be the source of AGW.

There have also been comments made about the possibility of the electric field above thunderstorms playing a role in Es, aided by recent interest in sprites and elves etc. A very useful article seems to discount these links as being ineffective above about 90km in the E region [11]. If the electric fields above thunderstorms were to be significant, then we would expect thunderstorms to feature strongly on Es days, but this is not the case.

In fact, after 30 years of looking, there have been very few that might have played a direct role, and these were very slowly-moving severe summer thunderstorms. The successive pulse effect of such extreme lightning-generated electric fields would not be sustained even if it had reached up to 100km and above. It is only fair also to mention some work [12] which described the negative effect of a thunderstorm on pre-existing Es. I am sure the thunderstorm involvement with Sporadic-E will not go away, if only for the reason that thunderstorms are commonplace in the summer months, but we may need to look at associated upper trough vorticity and turbulence more closely in the vicinity of severe thunderstorms. However, returning to the main point, it is the case that despite a lot of research attention, thunderstorms are not often seen in the right places when Sporadic-E is worked.

WEATHER: MOUNTAIN WAVES

This is a special case of the jet stream example where over many years of closely noting such events, a frequently-occurring case of the link between the weather and Es happens when a jet stream blows over a mountain range (Figure 15). The wind is deflected by the mountain or hill and the upward displacement can produce a vertically propagating AGW which, in turn, provides the necessary wind shear in the E-region to make Es or enhance a pre-existing tidal wind-shear produced Es layer. Europe and the United States, for example, are well-blessed with suitable mountain ranges and are frequently visited by overlying jet streams, so it



FIGURE 15: Mountain ranges in EU and NA.

Jet stream blows from	Region	Frequency
NNW or N	Alps	Common
N	Pyrenees	Common
NE	Balkans	Rare
W	Sardinia	Rare
SE	Norway	Rare

TABLE 1: Es and some mountain ranges.

is not surprising that many of the Es openings are associated with jet streams. I would say that, subjectively, the proportion of Sporadic-E events which fall into this category may be as high as 85 per cent. This does give an easy win for radio amateurs who want to evaluate the prospects for Es in a given direction; just focus on where the jet streams are located on the upper air charts in relation to the well-known mountain ranges.

Some mountain ranges are frequently mentioned in dispatches when Es is reported. For Europe, my suggestions are given in Table 1.

WEATHER: UPPER RIDGE PATTERNS

There is a well-known feature of surface TV weather maps known as a **ridge of high pressure**, which is an extension of an area of high pressure. Similar features can be found on upper-air charts, and in both cases can be marked as a line along which the direction of the flow changes from going up one side of the ridge to down the other. These are also known in aviation as regions of clear air turbulence (CAT) and tend to move quite slowly, so may be useful sources of continuous streams of AGW, but once again appear relatively uncommon as triggers for Es on the amateur bands.

When a ridge is very sharp, i.e. has a thin or narrow shape, then the stronger flow either side of the ridge axis may actually be the

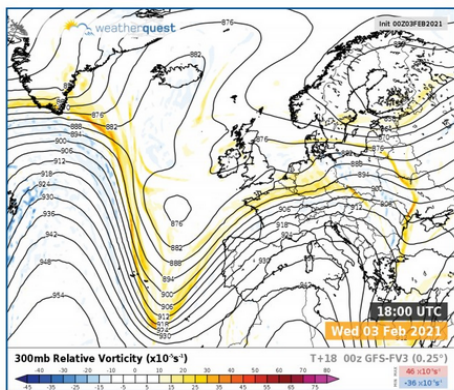


FIGURE 16: Relative vorticity.

real generator of any AGW. Indeed, it is not altogether resolved which part of a jet stream or upper ridge is the most likely source area and a common factor of many openings is that several of these potential triggers are often in close proximity. Therefore, there is plenty of work radio amateurs can do to try and pin down these sources of AGW by logging QSOs as they happen and relating them to the locations of the weather features capable of generating atmospheric gravity waves. My attempt at developing an algorithm for predicting Es associated with ridges of high pressure ended up with a representation of relative vorticity, or anticyclonic vorticity in the case of upper ridges, which we'll say a bit more about in the next section.

WEATHER: RELATIVE VORTICITY

This is a recent addition to allow calculation of an automated probability index for Sporadic-E,

16). Weather fronts are usually regions of high positive relative vorticity and sometimes upper ridges or cols (regions between highs and lows) show up well as maxima of negative relative vorticity or anticyclonic vorticity.

These relative vorticity values are included in an automated Es Probability Index (EPI) detailed in a later section about the Propquest website.

OPERATING ES: CLUSTERS AND SKIMMERS

The first tool in the armoury should be the vast array of DX clusters, including PSKReporter [19], which has the advantage of not requiring input from users to see band activity. These represent a great starting point for day-to-day evaluation of Es prospects. Many allow you to follow the movement of the areas of activity as well as the progression through the bands as the foEs increases.

On a given band, it often pays to be in at

described in a later section. The basic meteorology part of this is that relative vorticity is a measure of the amount of spin the air has relative to the Earth's surface, and it can give an indication of the likelihood of generating AGW.

You can imagine that low-pressure areas are regions of higher relative vorticity, defined as positive values, and anticyclones, or high-pressure systems, are regions of high negative relative vorticity. It turns out that the really high values are usually focussed over smaller areas than the large extent of low or high pressure regions (Figure

the beginning of an opening since we know the height of an Es patch descends over time. It is well worthwhile developing a set of favourite beacons as indicators on each band from 10m through to 2m in the usual Es directions; the greatest ranges will be associated with the highest Es altitudes.

Band maps are very useful if you run a skimmer or logging program, and there are many online clusters which can automatically put up displays of call signs from the various sources.

OPERATING ES: PROPQUEST

Propquest [1] is a relatively-new propagation website, which contains many useful tools for keeping ahead of Sporadic-E, and is still being developed. Use the website to follow the descriptions presented here. There are three main sections which will help you keep on top of an opening:

Es Blog: The first is the Es Blog which contains jet stream charts for the current day and, during the main summer season, a blog of where the mostly likely weather triggers can be found. It is not as complicated as it looks, but you should read the notes in the **About** section.

Es Probability Index: The second item to help you is found on the EPI tab. This is an early attempt to combine all the many contributions to Es into one index which can then be colour coded on a map. This takes a lot of the work out of tracking down all the many and varied elements in the Sporadic-E schematic described above. The value of EPI is given by the product of two functions:

$$EPI = f(\text{weather}) \times f(\text{nn}).$$

The weather term, f(weather), consist of things like jet stream, vorticity, rate of change

of contours of upper air pattern, convection parameter for thunderstorms, plus others, and shows the location of the relative chance of generating AGW. The adjustment term f(nn) is close to 1, and modifies the EPI according to things which are known to have a limiting or enhancing influence on Es like the Kp index, magnetic dip angle, meteor input, time of day, season, topography, and (probably) a solar factor. This section is under development and will include the ability to plot a path between locator squares to see how the path fits with hot spots of EPI. The timeline slider will allow you to move the display ± 6 hours to see how conditions have changed, or will change, so that you can evaluate the chances of further paths in the second semidiurnal tidal wind operating window of the day.

FOF2 GRAPHS: The third component can be found under the foF2 tab and shows the latest near real-time ionosonde values of several useful propagation parameters including the foEs. I recommend looking at the averages section to see how the time of maximum foEs varies from month to month. It is probably worth mentioning that Es can affect the LF bands too, and there is an example detailed in the About section of Propquest. Strong signals 80m, when foF2 and the Spread-x index (Fxl) are well below 3.5MHz, are most likely to be as a result of Es.

Much work remains to be done on this site, so please let me know if you have any suggestions for the wish-list. One current task is to build up experience of the value of the experimental EPI against maximum foEs to see if the algorithms are working or in need of adjustment. There is a need to evaluate the threshold between modes, i.e. the value of the EPI required before the propagation is strong enough to allow CW or SSB paths compared to

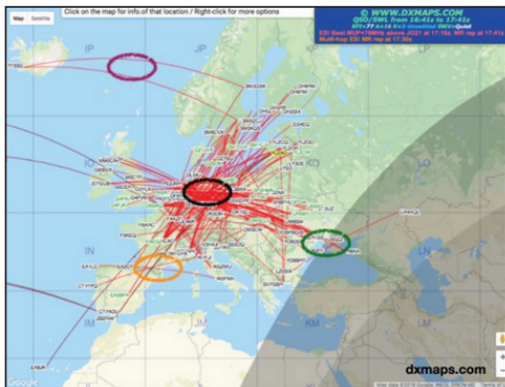


FIGURE 17: Single-hop Sporadic-E opening in Europe.

FT8, and a calibration of EPI value for the maximum foEs.

SPORADIC-E OPENINGS

Typical Sporadic-E openings usually start to show on 10m and then become more widespread as the ionisation intensifies. The effect on the signals you hear is that the skip shortens and this is a good time to test the next higher band, 6m. The typical distances range from about 800km up

to 2400km. Remember that it is the Es patch halfway between the two stations that you need to monitor. As mentioned earlier, the greatest ranges for single hop Es come at the beginning of an opening since the geometry becomes less favourable as the Es layer descends.

Losses for a single hop reflection are very low indeed, so strong signals are typical once the opening has started. However, the Es patch may be moving within the zone activated by the AGW from below. I like to think of it as a torch beam ellipse of AGW illuminating the E region, and within the beam ripples of Es moving through the ellipse. This means that the geometry between individual stations is transient, so short overs with report and locator are typical exchanges. Obviously, if it is a major opening with very strong ionisation, the paths may last longer. Don't forget to enter them into your cluster of choice so that others can follow

the developments. **Figure 17** shows a typical European opening, but note that quite often several separate single hop paths may be active during an opening.

Most Sporadic-E seasons involve talk of a few multi-hop paths to, say, the Caribbean, the United States, and even to Japan (**Figure 18**). These require stations with higher effective radiated power levels to offset the extra losses of multiple reflections. There is usually only a limited permutation of separate hops which fit a given path, and some reflections may have sub-optimal qualities.

A recent analysis of an early-summer opening to the Caribbean (**Figure 18**) focussed upon the climatology of the winds at jet-stream heights. These charts showed that the early summer upper air wind speeds were much higher for the third hop near the Caribbean in May and very much lighter in August (**Figure 19**). This

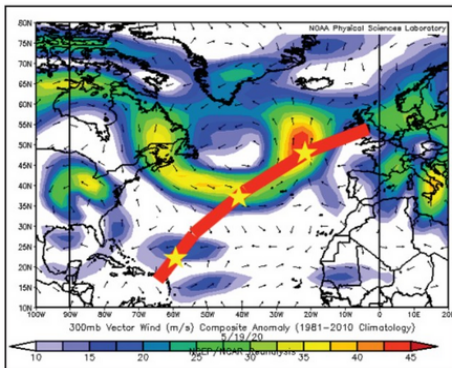


FIGURE 18: Multi-hop path to the Caribbean.

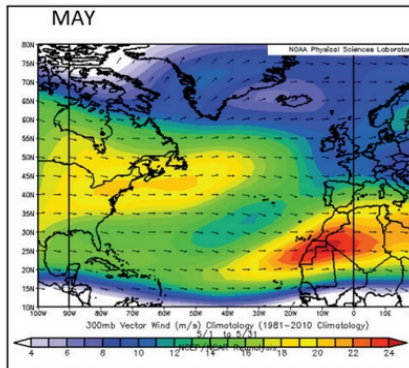
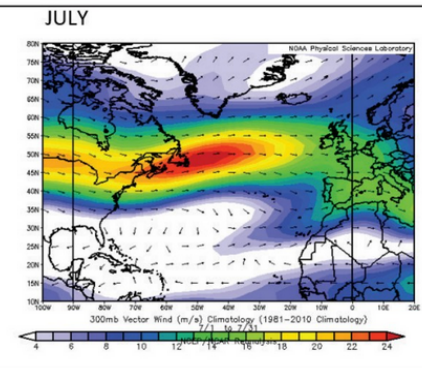


FIGURE 19: Mean Atlantic wind speeds in May and August.



suggests that Caribbean openings should be more likely at the start of the season than at the end. I suspect other paths may yield similar insights into their climatological likelihood.

There has been much discussion about paths to Japan in high summer, including reports of polar mesosphere summer echoes (PMSE). In some early tests of the EPI maps, I examined a route across northern Russia and down across China to Japan. The path, when plotted as a great circle, follows a route across northern Russia and down across China to Japan. The extended summer daylight hours herald the changing weather patterns where we see traditional wave depressions tracking across these northern latitudes, along with the jet stream activity associated with them.

The path investigated seems to be supported by the traditional multi-hop mechanism with good evidence of jet streams at the various points of reflection.

As I write this, with the Editor pressing me for copy, I have just spent some time distracting myself with an examination of my log for an Es QSO; the first one I found was on 28th June 2020 with an OH operator. This is not the best example, but it will serve to show how we can look for evidence of which semidiurnal tide is bringing our opening from easily-accessible ionosonde data [17]. It just goes to show how compulsive this propagation research really is!

Recalling the discussion earlier of the two semidiurnal descending tidal nodes, I wondered

whether we could see this on a representative ionosonde and allow the radio amateur to study the layer performance as it descends. The graph (Figure 21) plots the foEs and the virtual height of the layer, h'Es. You can see the morning descending node between 0600 and 1200UTC with a peak foEs at the lowest point. The second semidiurnal node can also be seen in this example, starting at 1600UTC and extending to 2200UTC. There are other upticks in foEs that don't fit this semidiurnal node approach, but they could be part of a diurnal tidal node. This will be investigated further during this year's Es season, and I can imagine a new tab appearing on the Propquest graphs.

WHERE NEXT WITH SPORADIC-E?

There are still parts of the Sporadic-E world which are either not well understood within the radio amateur community, or are sources of active research amongst the professional scientific community, and thus in a state of evolution. There is great value in discussion between radio amateurs and scientific researchers to share knowledge and generate a better-informed operating practice. HAMSCI [13] is a wonderful connection between these communities.

Radio amateurs can make a valuable contribution too, since we provide a fantastic resource of data for thousands of Es events over

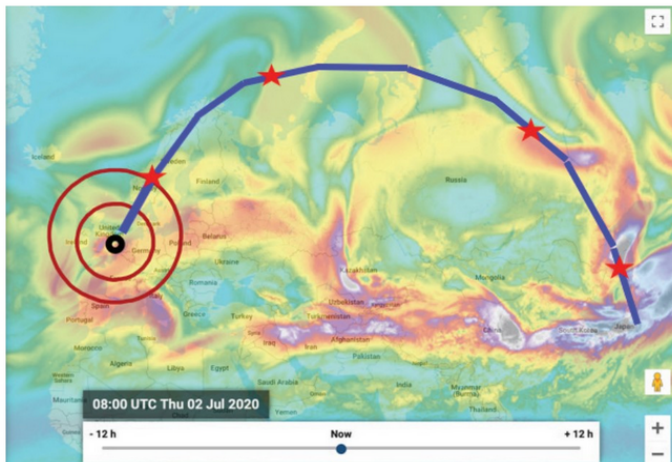


FIGURE 20: Multi-hop path to Japan.

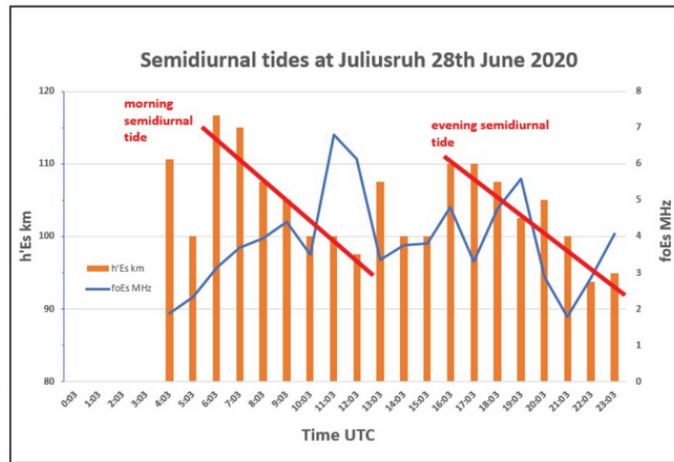


FIGURE 21: Juliusruh semidiurnal tidal nodes.

many decades, which can colour some of the climatology of Es. Recently, a very interesting paper by Chris Deacon, G4IFX et al. examined polarisation of Es signals using amateur radio paths [14]. In fact, since I first got involved in looking at Es in a series of articles in *RadCom* May-Aug1989 [15] numerous articles have been published in the amateur radio literature which contribute to our operating skill set. At the end of last year, Joe Dzekevich, K1YOW [16] examined the weather link to Es. A search of online presentations from various talks at conventions and radio clubs will reveal that the subject is widely discussed and continues to build the knowledge base; these should be part of your background reading.

Understanding the physics of Es is an essential part of improving our operating skills. It allows us to develop operating rules based upon what is actually happening rather than upon folklore. There is still plenty to learn and, after 30 years of looking at Es, I openly admit to big gaps in my knowledge. It is a most satisfying thing when new evidence emerges from amateur radio data. Many of our QSOs are in the fusion of short atmospheric gravity waves and the longer tidal waves and often go unrecorded in many research fixed-location MLT radar studies, which are not necessarily in the right place.

Before leaving this 'state of the nation' section I'll add a few items where I look forward to further knowledge or research being reported to fill in gaps (for me anyway); I suppose its really my wish list.

- 1) Which part of a jet stream produces the strongest atmospheric gravity waves?
- 2) Add new plot to show foEs and h'Es in Propquest.
- 3) Access the state of the semi-diurnal tides in real time to ascertain daily or seasonal

variability.

- 4) Is the variability of Es from one year to the next produced by changing weather patterns or meteor input or both?
- 5) Access real-time meteor input count.
- 6) How important is the solar flux or coronal mass ejection contribution?
- 7) How significant is the stratospheric wind regime in filtering AGW?
- 8) Can we derive a relationship between AGW or jet-stream strength and the foEs max?

There are bound to be others, but this list will do for now. Aside from the sheer joy of finding out that something actually works, the target for many radio amateurs is to develop an effective operating guide which explains the 'why, where and when' of Es. I hope this article will make your next Es season more successful or, at the very least, more understandable.

From all who work in propagation research, a big thank you for entering your Es QSOs into the various clusters – it really does make a difference! I am grateful to NOAA Physical Sciences Laboratory [18] for making reanalysis data available online as used in Figures 18 and 19. Finally, I would like to thank my colleagues at Weatherquest for providing resources for Propquest and Dan Holley for his software support.

WEBSEARCH

- 1: Propquest website, www.propquest.co.uk
- 2: Chu, Y. H., C. Y. Wang, K. H. Wu, K. T. Chen, K. J. Tzeng, C. L. Su, W. Feng, and J. M. C. Plane (2014), Morphology of sporadic E layer retrieved from COSMIC GPS radio occultation measurements: Wind shear theory examination, *J. Geophys. Res. Space Physics*, 119, 2117–2136, doi:10.1002/2013JA019437.
- 3: Whitehead, J. D. (1989), Recent work on mid-latitude and equatorial sporadic E. *J-Atmos Terr Phys* 51(5):401-424.
- 4: M.A. Abdu, D. Pancheva (eds.), A. Bhat-tacharyya (Coed.), *Aeronomy of the Earth's Atmosphere and Ionosphere*, 381 IAGA Special Sopron Book Series 2, DOI 10.1007/978-94-007-0326-1_29, © Springer Science+Business Media B.V. 2011.
- 5: Plane, John. M. C. (2003), *Atmospheric Chemistry of Meteoric Metals*. Chem. Rev. 2003, 103, 4963-4984.
- 6: Haldoupis, C., D. Pancheva, W. Singer, C. Meek, and J. MacDougall (2007), An explanation for the seasonal dependence of midlatitude sporadic E layers, *J. Geophys. Res.*, 112, A06315, doi:10.1029/2007JA012322.
- 7: Selvaraj, D., A. K. Patra, S. Sathishkumar, K. Kishore Kumar, and D. Narayana Rao (2017), On the governing dynamics of the VHF radar echoes from the mesosphere and collision-dominated lower E region over Gadanki (13.5°N, 79.2°E), *J. Geophys. Res. Space Physics*, 122, 1163–1177, doi:10.1002/2016JA023297.
- 8: Cierpik, K. M., J. M. Forbes, S. Miyahara, Y. Miyoshi, A. Fahrutdinova, C. Jacobi, A. Manson, C. Meek, N. J. Mitchell, and Y.

Portnyagin, Longitude variability of the solar semi-diurnal tide in the lower thermosphere through assimilation of ground- and space-based wind measurements, *J. Geophys. Res.*, 108(A5), 1202, doi:10.1029/2002JA009349, 2003.

- 9: Barnes C., GW4BZD Examining sporadic E RadCom Plus May 2015 22-31.
- 10: Tokyo Climate Center <http://ds.data.jma.go.jp/tcc/tcc/products/clisys/STRAT/>.
- 11: Haldoupis, C. (2018) Is there a conclusive evidence on lightning-related effects on sporadic E layers? *Journal of Atmospheric and Solar-Terrestrial Physics* 172 (2018) 117–121.
- 12: Searching for effects caused by thunderstorms in midlatitude sporadic E layers Veronika Barta, Christos Haldoupis, Gabriella Satori, Dalia Buresova, Jaroslav Chum, Mariusz Pozoga, Kitti A. Berenyi, Jozsef Bor, Martin Popek, Arpad Kis, Pal Benecz *Journal of Atmospheric and Solar-Terrestrial Physics* 161 (2017) 150–159.
- 13: <https://hamsci.org>.
- 14: Deacon, C, Witvliet, BA, Mitchell, C & Steendam, S 2020, 'Rapid and Accurate Measurement of Polarization and Fading of Weak VHF Signals Obliquely Reflected from sporadic E Layers', *IEEE Transactions on Antennas and Propagation*.
- 15: Bacon J. G3YLA 1989, *RadCom* May-Aug.
- 16: Dzekevich J. K1YOW 2020, *CQ Magazine*, Nov 28-32.
- 17: <http://giro.uml.edu>.
- 18: <https://psl.noaa.gov/data/composites/day/>.
- 19: <https://www.pskreporter.info>.

piWebCAT: a CAT software system for transceiver control

A trawl of the internet for amateur radio computer aided transceiver (CAT) software reveals a range of different applications. Many are available free of charge; some are for purchase. For effective control of a transceiver, a CAT system must provide:

- a frequency display with associated effective tuning control;
- a comprehensive range of receiver and transmitter switches and level settings, including band-switching and the ability to switch between transmit and receive (sometimes referred to as MOX);
- S-meter and power meter displays; and
- the ability to operate with a wide range of radios from different manufacturers.

The majority of the CAT software applications run under Windows on a PC, but there are also some that run under Linux. For most radios, CAT software connects to the radio via a USB or RS232 serial interface. Icom radios, however, use the Icom CI-V interface, which is

a bi-directional, "single-wire" interface using a 3.5mm mono jack plug.

CAT CONTROL USING A WEB BROWSER

A web-based CAT solution allows remote control of a transceiver from a web server via a local area network or the internet. The controlling device can be a desktop computer, a laptop, a tablet or even a mobile phone. All that is needed is an HTML5-compatible web browser such as can be found on almost any computer, tablet or smartphone. When I searched the internet for web-based CAT control, I found MFJ's recent MFJ-1234 system, and one other free application. In this article, I describe my piWebCAT application, which is a new (free) application with extensive user configurability and integrated station logging. It uses a Raspberry Pi as the CAT controller and web server. I refer to the web browser as the client, and to the LAN/WLAN link between server and client as the IP link.

USING A RASPBERRY PI

piWebCAT provides CAT control of transceivers with integrated station logging from an Apache2 web server running on a Linux Raspberry Pi 4B computer, using PHP7 server programming and a MariaDB (MYSQL) database for configuration storage. This combination is often found and is commonly referred to as the 'LAMP' web server configuration (Linux, Apache, MySQL, PHP).

The Raspberry Pi is connected to your radio via a USB interface, or via one of its serial ports on its 40-pin GPIO connector. You do not need any additional hardware if you use the USB connection. However, for RS232 serial connection, you need to add a low-cost RS232 daughter board. There is one available commercially (**Photo 1**). Alternatively, I can supply my serial interface card that also includes an Icom CI-V interface (**Photo 2**). (This is a bare board: you build it!)

An extensive manual can be obtained as a pdf document from [1]. Section 14.1 provides

Ian Sumner G3VPX
ian@g3vpx.net

the links for downloading the software and some configuration files.

Figure 1 shows the main piWebCAT client main window when the server is connected to my Yaesu FTdx101D. This window is in a web browser connected via the IP link to the web server on the RPi.

RECEIVE AND TRANSMIT AUDIO

The addition of an audio link to a web-based system enables full voice operation remote from the transceiver. I achieve this with Mumble [2]. Mumble is a bi-directional audio system that resides both on the RPi and on the client device. It operates alongside piWebCAT, but is separate from it.

Installation instructions are given in Section

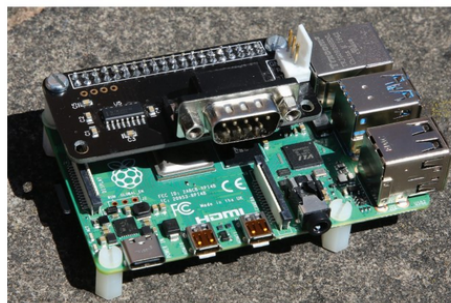


PHOTO 1: Raspberry Pi 4 with a commercial serial card.

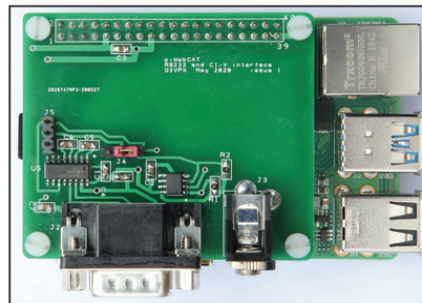


PHOTO 2: piWebCat with my own serial card.

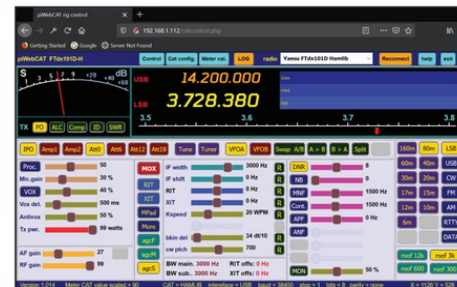


FIGURE 1: piWebCAT client main window.

ID	rig	description	color	caption	button	sector	code	action	vs	action	action	startoff	startoff	start	name	comment	
11	FTdx101D	Select 1st band	indigo	10	Y	BAK0	G	V	0	0	0	0	0	0	10	77	
12	FTdx101D	Select 2nd band	indigo	6m	12	Y	BAK0	G	V	0	0	0	0	0	10	77	
13	FTdx101D	Select 4th band	indigo	4m	13	N		U	0	0	0	0	0	0	0		
14	FTdx101D	Select 2m band	indigo	2m	14	N		G	0	0	0	0	0	0	0	0	
16	FTdx101D	Select 3rd band	indigo	3rdm	16	N		G	0	0	0	0	0	0	0	0	
17	FTdx101D	VFO A operation	magenta	VFOA	16	Y	VFO	G	X	0	0	0	0	0	0	0	=> big MHz button on rig
18	FTdx101D	VFO B operation	magenta	VFOB	17	Y	VFO	G	X	0	0	0	0	1	1	0	=> big SSB button on rig
23	FTdx101D	Receiver wide	teal	reaf 12h	31	Y	ROOF	G	V	0	0	0	0	0	1	0	
24	FTdx101D	Receiver SSB	teal	reaf 3h	32	Y	ROOF	G	V	0	0	0	0	0	2	0	
25	FTdx101D	Receiver CW	teal	reaf 60h	33	Y	ROOF	G	V	0	0	0	0	0	4	0	
26	FTdx101D	Receiver narrow	teal	reaf 30h	34	Y	ROOF	G	V	0	0	0	0	0	5	0	
27	FTdx101D	MOX	#000000	MOX	59	Y	MOX	T	X	01	02	00	00	0	0	0	
28	FTdx101D	Tune auto	indigo	Tune	75	Y	TUNE	T	X	2	2	2	2	2	0	0	
29	FTdx101D	Tuner autooff	indigo	Tuner	76	Y	TSSB	T	X	1	1	0	0	0	0	0	
32	FTdx101D	Copy VFO A to B	#000000	A > B	80	Y	ATOB	S	X	0	0	0	0	0	0	0	No button on rig for this ?
33	FTdx101D	Copy VFO B to A	#000000	B > A	81	Y	BTDA	S	X	0	0	0	0	0	0	0	No button on rig for this ?

FIGURE 2: Buttons configuration editor table.

14 of [1]. Mumble-server and mumble (client) are installed on the RPi, and a mumble client app is installed on the controlling device (eg Mumia from the Android play store). There are also some other Voice Over IP (VOIP) systems that could be used instead of Mumble.

CONFIGURING PIWEBCAT FOR TRANSCIVERS

Most transceiver manufacturers provide details of how to use their CAT interfaces, either in their operation manuals or in separate CAT manuals. This information is needed by the authors of CAT software, but mostly not needed by the users of CAT software. However, piWebCAT's initial design was different in this respect in that it provided a means of configuration which was different for each transceiver. I have provided configuration files for text-based CAT radios such as newer Yaesu and Kenwood transceivers, older binary-data based Yaesu rigs, and for Icom CI-V based radios. However, piWebCat also allows

the use of Hamlib [3] that supports some 250 radios, and some rotators.

Hamlib's rigctl interface accepts commands in its own generic command set [4], which are then translated into the appropriate CAT commands for the selected rig. You select your radio with a reference number. The user may still need to do some fine tuning of one of the existing configurations, but this is a lot less work.

Unlike with most other CAT software, piWebCat allows the user to configure controls for the many parameters located in transceiver configuration menus. This feature also applies to Hamlib because piWebCat provides a means to enter rig-based commands which are not in Hamlib's generic command set.

CONFIGURATION

piWebCAT's user interface display is populated with 27 slider controls, 66 buttons and 24 extra buttons on a popup window. This arrangement is fixed; however, you can change the function

of most of the buttons and slider controls. For example, you can change captions and colours, you can organise buttons as toggle switches, or single-action switches, or group them to a common function. You can configure a button as an 'LED' indicator which changes from black (OFF) to a bright colour of your choice (ON). (See Figure 2, the buttons-editor table.) If a particular control is absent, you can create it using a 'spare' control or an assigned control that you don't need.

SUPPORT

As already mentioned, the manual and software downloads are available from [1]. However, the full software download, as a zipped Raspbian Linux SD card image, is 3.5GB in size and may take a very long time! My current policy is to supply a pre-written SD card by mail at cost. Using this SD card, piWebCAT should be almost 'plug and play' for some radios, and require only a little setting up for others. (If demand is higher than anticipated then I might upgrade my web hosting to allow SD card image downloads!).

DEVELOPMENT

I began work on the project in December 2019. I already owned two Raspberry Pi 4 computers. I purchased a Serial Pi Zero board (under £10; see Photo 1 and Websearch [5]). This small board has an RS232 interface chip and a standard 9-way serial connector. It mounts on the 40-way GPIO connector of the RPi. The radio was a newly-purchased Yaesu FTdx101D.

I saw two areas of the design that were critical to the success of the project. These were (1) provision of a good S-meter presentation, and (2) an effective tuning mechanism. Both were potentially not easy to achieve on a web page. Of particular concern was the challenge of achieving

a sufficiently-rapid updating of transfers between the client and the radio. The S-meter needed frequent updates from radio to client in order to respond to a rapidly-changing signal. The tuning mechanism needed a rapid data repetition rate between client and the radio to achieve a useable smooth tuning action.

The data speeds from PC-based CAT software are only limited by the serial link's baud rate and brief delays in the radio. There are also other delays when using a web server which has a more-complex data path between it and the client.

I intended from the start that the project would have to be completely user-configurable for multiple radios, and that the configuration would be stored in a MYSQL database on the RPi server. I have now achieved this. However, for initial feasibility testing there was no database, and I simply hard-coded the S-meter and frequency-control commands for the FTdx101D.

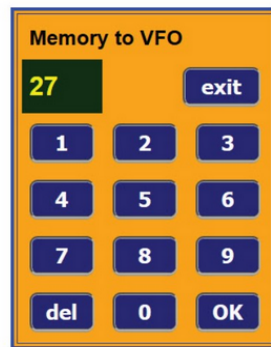


FIGURE 3: Memory-select keypad popup window.

The first decision was the size of the web page on the client. I chose a size of 1024 x 558 pixels, which would fit nicely on a 1200 x 800 android tablet. The popup logging window, developed later, was made adjustable in size and is able to fit either alongside or below the main window on a standard 1920 x 1080 HD monitor.

There are two other popup windows: a memory selector keypad (**Figure 3**) and a window with 24 extra user-configurable buttons (**Figure 4**). The positions of these popup windows can be changed using a mouse (or a finger).

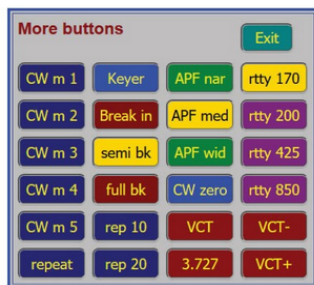


FIGURE 4: More user-configurable buttons popup window.



FIGURE 5: S-meter.

The S-meter background was (and still is) a 250 x 110 pixel bitmap. It was modelled on the FTdx101D with black background and white and blue markings. After some experimenting, I settled for a single curved scale with a radius larger than the radius of movement of the virtual red meter needle. The meter needle had to appear to be rotating around an axis which was at some invisible point below the bottom edge of the bitmap. This was tricky and involved a lookup table for the coordinates of the ends of the needle. Once this was done, the meter could be made multi-purpose by simply changing the background bitmap (see **Figure 5**)

I designed a linear tuning scale for each band to fit the available width of the tuner (see Figure 1). There are built-in tuning scales for the bands from 160m to 70cm. The tuning scale has a red marker which indicates the current frequency. The resolution is within 1kHz on 80m. Coarse frequency changes within the band can be made by simply clicking the tuning scale. Fine tuning can be made using the mouse thumbwheel, or by dragging the mouse horizontally within a blue tuning strip. Three tuning strips allow slow, medium and fast tuning by setting different Hz/pixel tuning rates. The mouse wheel tuning rate is also slow, medium or fast according to which strip the mouse pointer is in. (It defaults to medium if the mouse is elsewhere.) The two sets of tuning rates are user-configurable in the database.

With this S-meter and tuner design, I was able to demonstrate workable update rates for S-meter operation with concurrent tuning action. At this point, I shelved the project to work on completing my EncoderCAT project, but then came Covid-19 lockdown and lots of free time! The final result is illustrated in Figure 1.

piWebCat features include the following:

- extensive, flexible user configurability for

multiple radios;

- configuration stored in a MYSQL database on the RPI server;
- connection to Raspberry Pi 4B computer USB port or RS232/Icom CI-V GPIO card;
- downloadable, preconfigured, micro SD card image;
- documentation on a website, help screens and a pdf manual;
- tuning as described above;
- up/down buttons user settable (eg $\pm 12.5\text{kHz}$);
- fourteen bands, 160m to 70cm, each with a dedicated tuning scale;
- 66 buttons, many of which are re-configurable;
- 24 additional configurable controls on a popup window launched from a main-window button of your choice; and
- 27 re-definable slider controls.

Each of the slider controls has an adjacent value display with optional scaling, decimal point, lookup and units. Nine have an adjacent on/off button. Nine have a reset-to-default button. Many buttons are freely programmable and may be configured as 'LED' indicators with a bright, user-selected colour.

At start up, where possible, all controls are set to their values in the radio (where appropriate: for the current VFO). Selected controls can be configured to stay in sync with their associated parameters on the radio. (Repetitively automatic syncing of all buttons might use too much CAT bandwidth.)

The S-meter includes five button-selectable transmit meter options. There are six corresponding meter scale backgrounds. (These could be replaced by the user.) Meter accuracy can be enhanced using optional 20-point calibration tables with linear interpolation.

Memories (frequency, mode etc) must be programmed on the radio but can be numerically selected on piWebCAT. Any button can be allocated to a specific memory number and can have your choice of labelling. In addition, there is a numeric memory keypad (Figure 3). This is launched by your chosen main-window button.

LOGGING SYSTEM

An integrated logging system is provided as a popup window (**Figure 6**). The log is stored in a MYSQL database table in the RPI server. The log can be exported as a .csv file. A button is provided in the main window that will launch

cno	date	start	finish	frequency	mode	power	callsign	sent	recd	label	locator	remarks
1568/2020	11-21	11:33	3.720800	LSB	400		G9WVE					Albert Warrington on IC7300
2707/2020	18-08	19:29	3.745000	LSB	100		G9IHH	57	59			Jim M Cahewick
28/07/2020	00:28	00:28	7.430000	LSB	100		G9HRE	59+	59+			John in Newton On F920
23/07/2020	12:57	13:19	3.738380	LSB	100		G9LUN	59	59+10	BB		Terry in Bolton Discussion about Lancashire dialect!
20/07/2020	10:32	10:50	3.743730	LSB	100		G9VWI	57	58			Frank in N. of Chesterfield
20/07/2020	10:14	10:28	3.743560	LSB	100		G9VVK	57	59			Andrew in Bedford
10/07/2020	16:28	16:40	3.730450	LSB	100		G9DDC	59	59			Martin in Blackpool - remote operation contact using remote Fex rig

FIGURE 6: Station log window.

an Excel spreadsheet which can be printed. In operation, you first enter the call sign in the box provided. A single button click then lists previous QSOs with this station. Auto-entry buttons are provided which assign the date, start-time, frequency, mode, and power settings of the radio, and the previously-entered call sign. The 'finish' button can be clicked later to assign the end time of the QSO.

The log has an optional, auto-incrementing contest number field. A QRZ button does an immediate QRZ.COM lookup. Searching on any of the fields is provided.

EDITING THE CONFIGURATION DATABASE

piWebCAT provides a spreadsheet-like grid for each of the database tables (Figure 2). The grid uses phpGrid software from China for which I have an OEM distribution licence. Editing is directly on the grid. Several data fields use a drop down list selector in the grid cell. This includes a list of radios on the system which is automatically compiled from the rigs table.

The database tables can be exported to Excel and thereby printed. External, PC-based database editors can be used. I suggest the free 'MYSQL Front' [6] that can also export SQL files for backup or exchange of configuration data sets between users. This can also backup the log.

CAT INTERFACES

piWebCAT supports three serial data options: RS232, Icom CI-V, and USB from an RPi USB port. There are four configuration options: ASCII, YAESU5, CIV and HAMLIB. These options are selected in the rigs database table.

TABLETS AND PHONES

piWebCAT works well on my fast, Lenovo, Android 7, 10-inch tablet. Tuning is by

horizontal finger drag on the tuning area, although this is not quite as good as with a mouse. The mouse also adds thumbwheel tuning. My much-older android tablet is a little slow. The elderly mini-iPad works fine apart from displaying oval buttons which are slightly overfilled by their captions.

I solved one potential problem with Android related to dragging. We have a finger-drag tuner, but finger dragging on an Android device usually moves the whole window!

HARDWARE

piWebCAT has run on a pi Zero, but it was somewhat slow. I suggest using a RPi 4B with sufficient memory installed: each of mine has 4 Gbyte RAM.

As stated previously, the Serial Pi Zero board is an adequate, low-cost RS232 interface. However, if you want to interface to an Icom radio, you will need my piWebCAT board (Photo 2). This has both an RS232 connector and a 3.5mm mono jack for Icom CI-V. The driver circuitry is identical to that in my EncoderCAT project (July 2020 RadCom).

The Icom driver device is a reversed NXP PCA9600 device. This gives a transmit data output logic-low level of around 0.72V, which is ok for the radio's input circuit but which is not fed back to receiver data input on the RPi. The RPi coding does not anticipate Icom 'echo'. Some early RS232 CI-V converters may therefore not work. Hamlib copes with or without an echo. (A simple PCB mod can reinstate the CI-V echo if required.)

I have had 100 piWebCAT PCBs made in China and can supply them for £4 including postage and packing. You will need to buy the components and solder them onto the board yourself. The piWebCAT PCB has the same

physical width as the RPi, and so has the advantage that the connectors can be flush with the side of a box. You do not need either of these serial interface cards if using a USB connection between the piWebCAT and the radio.

piWebCAT and EncoderCAT can be used together if required with a single CAT connection to the radio. EncoderCAT acts as a router for piWebCAT serial data. (This does not work with HAMLIB). The EncoderCAT PCB uses a Baud rate of 15200 to link directly to the Raspberry Pi's serial port on GPIO pins 8 and 10. The 'rigs' table in PiWebCAT has a 'connection' field that can be 'RIG' to connect directly to the radio, or 'ENCAT' to connect via EncoderCAT. The ENCAT option sets a baud rate of 115200 and sends data packets to EncoderCAT with a header that informs EncoderCAT how many response bytes to expect.

DOCUMENTATION

The piWebCAT pdf manual has a number of sections including: 'Introduction', 'Configuration' with detailed information for ASCII, Icom CI-V and the older 'YAESU5' radios, 'How it works' giving an explanation of what happens (without coding details), 'piWebCAT code' providing a course on Ajax/MYSQL client-server communication, 'Setting up a Raspbian micro SD card for piWebCat' (but you don't need to do this if you use the SD card image), and 'Installing the Mumbles audio link'.

I have set up a user group at piwebcat@groups.io. Please ask questions, add comments etc there.

WEBSEARCH

1. www.piwebcat.g3vpx.net
2. <https://www.mumble.com/mumble-download.php>
3. <https://hamlib.github.io/>
4. <https://manpages.debian.org/unstable/lib-hamlib-utils/rigctl.1.en.html>
5. see [eg https://thepihut.com/products/serial-pizero](https://thepihut.com/products/serial-pizero)
6. <https://mysql-front.software.informer.com/6.0/>

Developing a Wireless Temperature Sensor

For some inexplicable reason I've always been fascinated by ways to control central heating systems and thought the majority of offerings by manufacturers, until recently, were rather crude. Over the years, I've dabbled with different systems with varying degrees of success. However, about a year ago I decided the time had come to have another go, mainly because my Honeywell wireless thermostat's display had faded (see **Photo 1**). The replacement cost of the thermostat was £115, which was more than I wished to pay, and I thought I could make something better for a lot less. For some time, I'd been toying with the idea of using a Raspberry Pi running a web server over Wi-Fi, with a battery-powered

wireless temperature sensor sending data to the Pi. With this basic structure in mind, I set off on my journey to undercut the big boys with my new improved thermostat!

The first part of the development, which I started in September 2019, focused on the wireless sensor. The development of the sensor turned out to be more challenging than I had originally envisaged, and it exposed a number of engineering difficulties related to power management, physical packaging, and the choice of processor. Wireless communications with the sensor were made with an NRF24L01 transceiver module that could be picked up on eBay for one or two pounds. These devices send and receive packet data on the 2.4GHz

band, the same as used for Wi-Fi, and are designed to be controlled by a microcontroller using a three-wire Serial Peripheral Interface (SPI). There are many examples on the Web of using these devices with Arduinos. However, for my application an Arduino-based sensor was out of the question because of the limited power available.

The first thing to be decided was the power source and, after studying battery tables, I decided that two AA batteries would probably be sufficient to power the sensor for at least a year. Tables suggested that a single cell AA alkaline battery had a capacity of roughly 2 Ampere hours. However, this figure assumed an end-point discharge voltage of about 0.9V, and the battery performance changed depending upon the load. For standby power consumption, I decided upon a power budget of 100µA giving a battery life of more than a year. Another design decision concerned how often the sensor would need to transmit temperature data to the receiver. As I was dealing with room temperatures, I concluded that an update rate of around once a minute would be adequate. The NRF24L01 required a supply voltage of between 2V and 3.5V, making it suitable for battery operation, and its data sheet noted that the module consumed about 10mA at a power output of 0dBm. Although on transmit its power consumption is quite high, the transmission period is less than 1ms (as explained below).

The choice of power supply immediately constrained the electronics to a working voltage of around 2.5 to 3 volts. In addition to the type of battery cell to be used, some thought had to be given to packaging the sensor. I had a specific small commercial enclosure in mind that had been designed for a temperature or humidity sensor. Unfortunately, I wasn't certain

Alan O'Donovan, G8NKM
alan.odonovan@btinternet.com

that it would be large enough to accommodate two AA batteries as well as the electronics. Scaling down to AAA batteries would alleviate this difficulty but would also reduce the energy available. In the end, I decided to take a pragmatic approach; stick with two AA batteries and, should a larger enclosure subsequently be required, then so be it. The choice of using two AA batteries was a chicken-and-egg problem as, until other areas of the design had been explored, I wasn't sure of the overall power requirements. Still, you've got to start somewhere!

For the microcontroller, I initially thought of using the ATmega8A microcontroller, mainly because I had a number of them lying around. Unfortunately, upon reading the processor's application notes, I soon realised that it wasn't suitable for battery operation. Its minimum operating voltage was 2.7V with a standby current of around 1mA. After studying a number of data sheets, I eventually decided to use the Microchip ATtiny48, a newer design of controller with a lower operating voltage of 1.8V and a standby current of around 10µA. Selecting this controller was certainly a case of choosing the right tool for the job. The ATtiny48 has only 4K bytes of programmable memory and 256 bytes of static ram. That's not a lot of memory, but I thought it should be enough for the job. The cost of the chip was around 80 pence and I bought a few of them from RS Components [1]. (I later discovered that RS Components also sold the ATtiny88, which is the same controller but with twice the memory,



PHOTO 1: Honeywell Thermostat with faded display.

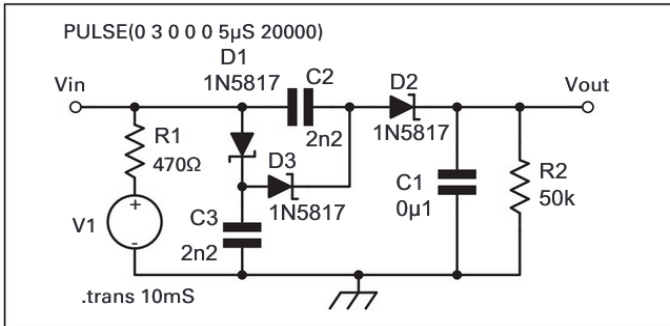


FIGURE 1: Spice model of the charge pump; note the use of Schottky diodes.

yet was 1 pence cheaper – huh!) Apart from the low power requirements, the ATiny48/88 has on-board SPI hardware (needed to support the transceiver module), a 10-bit analogue-to-digital converter (ADC) and a pulse-width modulator (PWM). However, it doesn't provide an on-board Universal Asynchronous Receiver Transmitter (UART), which would have been useful to enable a serial link to be set up between the processor and a PC for debugging purposes.

There are a number of devices available to measure temperature. I eventually settled upon the TMP37, a simple 3-pin solid-state device providing an output of 20mV/°C with an accuracy of $\pm 1^\circ\text{C}$. This device consumes less than 50 μA , but has a minimum operating voltage of 2.7V, which is a problem when the supply voltage range is 2.5 to 3 volts. To overcome this, I used the controller's pulse-width modulator (PWM) function to drive a charge pump that lifted the TMP37 supply

voltage to around 4.5V. The charge pump is based upon a voltage-tripler circuit that I modelled using LTSpice to understand its behaviour. The Spice model circuit diagram is shown in **Figure 1**, and **Figure 2** shows the input and output voltage over a timed period when the generator provides an excitation square wave of 2.8V at 100kHz. As can be seen, the output reaches full voltage after about 3ms. This analysis helped me to determine the delays to include within the microcontroller's software. In the real world, the circuit actually produced slightly more voltage than predicted (dependent upon the load). I reduced the voltage by changing the mark/space ratio of the PWM signal slightly, so as to decrease the energy transferred to the circuit.

In addition to transmitting temperature, there is also a need to monitor the state of the batteries, as ultimately the heating system relies upon the sensor to work properly. Without going into the minutiae of how this is achieved,

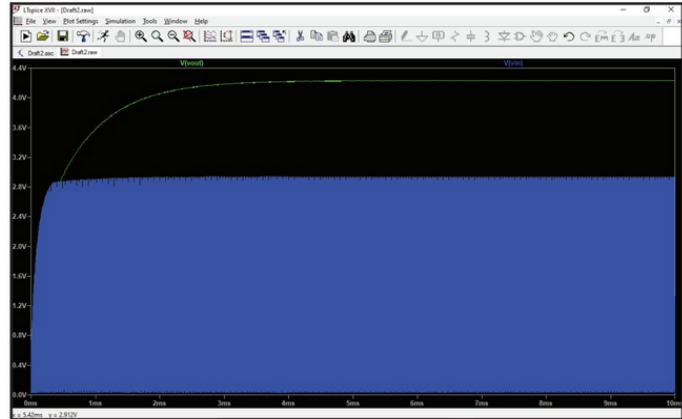


FIGURE 2: Spice model analysis; blue represents the supply voltage, and green represents the output voltage.

but using a bit of skulduggery involving the processor's ADC, internal voltage reference and various register settings, it is possible for the processor to measure its own supply voltage. I take no credit for the convoluted code necessary to achieve this trick as it is described in a Microchip application note [2].

I developed the software using a prototype ATmega8 processor with a serial link to a PC (see **Photo 2**). The prototype for the final "production" unit used the ATiny48 (**Photo 3**). As can be seen from the circuit diagram (**Figure 3**), there's very little electronics in the final design as nearly all functionality is carried out in the ATiny48 and the NFL24L01 modules. Note that no crystal is used for the processor's oscillator as the internal clock is used, nominally running at 8MHz and then

divided down to 1MHz to save power. The standby power consumption of the final circuit measured about 30 μA .

Application software was written in the C programming language using the ATMEL Studio 7 development system running on a Microsoft Windows 10 laptop. The compiler and development environment can be freely downloaded from the Web [3] and I found Studio 7 pleasant to work with as it includes a number of useful developer tools. To save power, the software makes use of the ATiny48's watchdog timer. This is an independent hardware timer built into the microcontroller. In the design, under normal circumstances the processor is in sleep mode with the main clock idle but power to the static memory and register memory maintained.

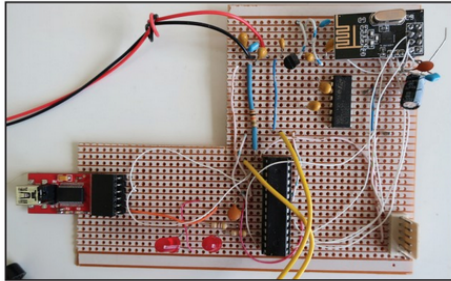


PHOTO 2: ATmega8 based prototype receiver.

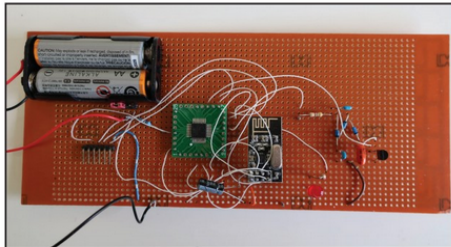


PHOTO 3: Prototype sensor.

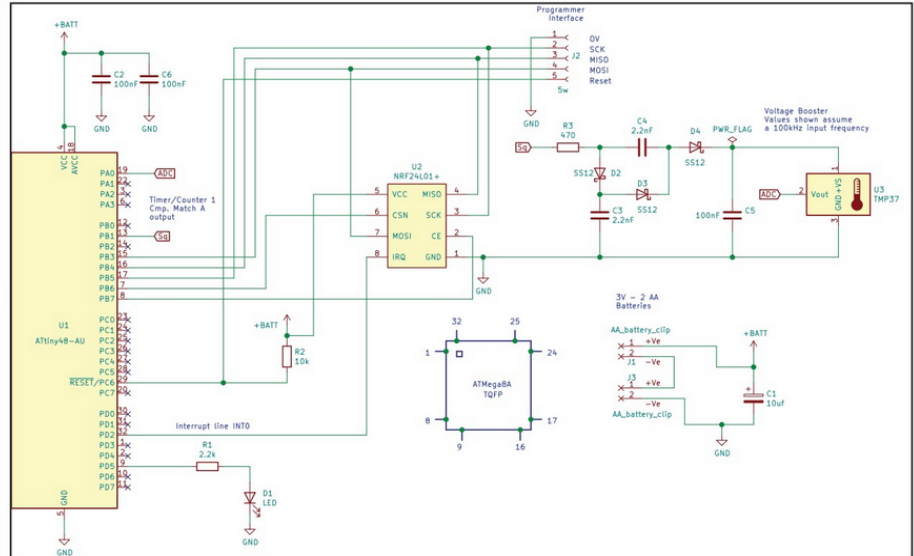


FIGURE 3: Final temperature sensor schematic diagram.

This ensures that the processor context is not lost when the processor reverts to sleep mode. When in sleep mode, the watchdog timer is active, and after 8 seconds (which is the longest delay the timer can manage) it fires an interrupt that wakes the processor. The software counts off eight interrupts and then the system is set into operation. In operational mode, the PWM is fired up, which in turn powers the TMP37 temperature sensor. The TMP37's output voltage is then measured by

the ADC, followed by the controller measuring the battery supply voltage. The resultant data are sent to the NRF24L01 module and are then transmitted having had a 16-bit Cyclic Redundancy Check (CRC) added. The NRF24L01's data rate is 1M bit/second that, with various transmission overheads, results in a data packet that takes about 700µs to send. Once the data has been sent, the NRF24L01 reverts to receive mode and awaits an acknowledgment (ACK) from the receiver. On

receipt of the ACK, the state of the NRF24L01 interrupt line changes and this signal is used to interrupt the microcontroller, putting the system back into sleep mode. This procedure is repeated ad infinitum. The memory footprint for the ATtiny48 wireless sensor was 1450 bytes. It's amazing what can be accomplished with such small programs when working at this low level.

The two breadboards shown here were used as transmitter and receiver to test out the

software system and to overcome the many difficulties along the way to produce a working system. As I mentioned at the beginning of this article, it was my intention to base the heating system upon a Raspberry Pi. To this end, I connected an NRF24L01 module to a Raspberry Pi and translated the ATmega8 C receiver code into Python (which seems to be the programming language of choice for folks who tinker with these devices). I had never used Python before but it seemed like a "breath

of fresh air” compared to “hacking” around with C on the ATtiny microcontroller. **Figure 4** shows the Pi’s output when the sensor is transmitting data. I’ve included a sequence number, generated by the sensor and incremented upon each successful transmission.

I laid out a circuit board for the sensor using KiCad [4] that I then had manufactured at one of the usual Chinese PCB prototyping factories. Using surface mount components to keep the size of the board to a minimum and using printed circuit board battery clips, I managed to squeeze the board into the small plastic enclosure I initially wanted to use (see **Photo 4** and **Photo 5**). The sensor has been on soak test for the past nine months, over which time the battery voltage has dropped by around 70mV. During the past few months, I’ve made a couple more sensors and all have worked as expected. All in all I am pleased with the results of this project.

I’ve placed the code on GitHub [5] for those who may be interested in viewing it.

WEBSEARCH

- 1: RS Components – <https://uk.rs-online.com/web/>
- 2: Microchip application note AN2447
- 3: <https://atmel-studio.software.informer.com/7.0/>
- 4: KiCad – Electronic Design Automation Suite <https://kicad.org/>
- 5: <https://github.com/alan-od>



PHOTO 4: Internal view of finished sensor boxed in a commercial enclosure.



PHOTO 5: Sensor in situ.

```
pi@raspberrypi: ~/python/mx_v1
File Edit Tabs Help

Fri May 22 18:07:08 2020
Unit Number = 1
Battery Voltage = 3.1
Temperature = 24.0
Sequence number = 154

Fri May 22 18:08:14 2020
Unit Number = 1
Battery Voltage = 3.1
Temperature = 24.0
Sequence number = 155

Fri May 22 18:09:20 2020
Unit Number = 1
Battery Voltage = 3.1
Temperature = 24.0
Sequence number = 156
```

FIGURE 4: Pi screen shot; it was a warm evening!