Version 1.0 March 2020 Gwyn Griffiths gwyn@autonomousanalytics.com

WSPRDAEMON Timescale Database Notes

Commands in this guide are shown in red in Courier font, and responses in blue. All the testing has been on a remote Raspberry Pi running Raspian 'Buster' unless otherwise noted.

0. Preparation at the client

Read access to the wsprdaemon Timescale spots database, at this stage, take two main forms:

1. Using a PostgreSQL installation on your local computer or
2. Using a one-line command line bash script that in turn uses PostgreSQL and Python utilities that need to be installed on your computer.

0.1 Install PostgreSQL on a Raspberry Pi

Note: If you already have an SQL program installed, e.g. SQLite, MySQL there may be an issue when it comes to installing PostgreSQL - unfortunately these problems may only come to light as an installation is attempted.

```
sudo apt install postgresql libpq-dev postgresql-client postgresql-client-
common -y
```

To read from and write to the wsprdaemon Timescale database you will also need an adapter program psycopyg2 for Python3. You will of course need Python 3 installed, and you may need pip3 if not already installed

```
sudo apt install python3-pip
sudo pip3 install psycopg2
```

For installation on other operating systems see https://www.postgresql.org/download/

0.2 Bash script interface

Obtain the wdquery_ts.sh script from Gwyn Griffiths (gwyn@autonomousanalytics.com)

Check that you have gawk installed, if not, install, e.g., for a Raspberry Pi:

```
  sudo apt-get install gawk
```

Note that you do not need Timescale DB extensions on your own machines, unless you intend to create your own Timescale databases.

This guide can only provide the bare essentials on using postgreSQL to query the wsprdaemon database. Full details on postgreSQL are available online[1] and at a very useful simplified tutorial website[2].

1. Gaining access

We have set up a universal read-only user id: wdread with password JTWSPR2008

With postgreSQL installed on your computer you can access the wsprdaemon database using:

```
psql -U wdread -d tutorial -h wsprdaemon.org
Password for user wdread: JTWSPR2008
tutorial=>
```

The -d option connects us to Database tutorial, this database has two Tables: spots and noise. The postgreSQL prompt shows the database name, and we will use the table name as needed in the from query.

Sections 2–4 of this guide cover the structure and use of the database in this mode. Section 5 covers its use via wdquery_ts.sh, a purpose-written command line script.

2. Timescale database structure

2.1 Spots Table

To list the columns within the spots table, with data types enter

\d spots

[1] See https://www.postgresql.org/
[2] https://www.postgresqltutorial.com/

The main output lines will show

```
                           Table "public.spots"
  Column   |            Type             | Collation | Nullable | Default
-----------+-----------------------------+-----------+----------+---------
 time      | timestamp without time zone |           | not null |
 band      | text                        |           |          |
 rx_grid   | text                        |           |          |
 rx_id     | text                        |           |          |
 tx_call   | text                        |           |          |
 tx_grid   | text                        |           |          |
 SNR       | double precision            |           |          |
 c2_noise  | double precision            |           |          |
 drift     | double precision            |           |          |
 freq      | double precision            |           |          |
 km        | double precision            |           |          |
 rx_az     | double precision            |           |          |
 rx_lat    | double precision            |           |          |
 rx_lon    | double precision            |           |          |
 tx_az     | double precision            |           |          |
 tx_dBm    | double precision            |           |          |
 tx_lat    | double precision            |           |          |
 tx_lon    | double precision            |           |          |
 v_lat     | double precision            |           |          |
 v_lon     | double precision            |           |          |
```

We will use these column names in our queries, most are self explanatory to WSPR users. The others are

- c2_noise is a legacy and is not used in this table.
- rx_az is the azimuth at the receiver of the incoming signal, this is not the azimuth shown at wsprnate.org, which is at the transmitter, tx_az
- v_lat and v_lon are the vertices, the position of the northernmost or southernmost point along a great circle path between receiver and transmitter.

2.2 Noise Table

For the noise table the structure is

```
                          Table "public.noise"
  Column    |           Type           | Collation | Nullable | Default
------------+--------------------------+-----------+----------+---------
 time       | timestamp with time zone |           | not null |
 site       | text                     |           | not null |
 receiver   | text                     |           | not null |
 rx_grid    | text                     |           | not null |
 band       | text                     |           | not null |
 rms_level  | double precision         |           |          |
 c2_level   | double precision         |           |          |
```

The site column is generally the callsign, or main designator, and the receiver a text string usually describing a specific receiver and or antenna configuration. To find out the receiver names in use at a specific site, follow this example for KPH, remembering the single quotes and the semicolon. Distinct means the query only returns one instance for each receiver.

```
select distinct receiver from noise where site = 'KPH' limit 10;
    receiver
----------------
 KPH_LF_72
 KPH_HF_78
 KPH_Kiwi_72_LF
 KPH_HF_77
(4 rows)
```

For further details of the receivers you would need to contact the operator of that site.

2.3 To end a session

```
> \q
```

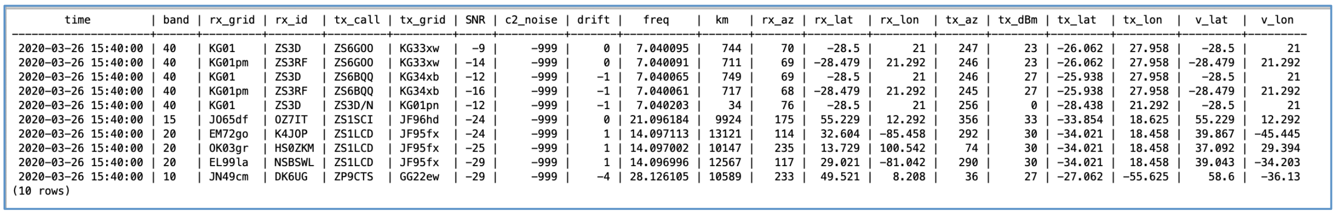
Sections 3 and 4 contain examples of interactive queries on the Timescale PostgreSQL database. Skip ahead to Section 5 if all you need is a simple Linux command line script with common list options.

3. Querying spots and noise data from the wsprdaemon database

This section provides examples of interactive queries following on from the access and database selection details in Sections 1 and 2. To see a short example, for the last 10 records in the database with the newest first 'order by time desc' is included in the command, and the limit 10 sets how many records to output to the screen. The * signifies all fields. The semicolon at the end of the line is essential. It signifies the end of the query, as queries can span several lines with just returns. If you forget, just type ; at the next prompt.

First, connect to the database:

```
sudo -u wdread psql -h wsprdaemon.org
password: JTWSPR2008
tutorial=> select * from spots order by time desc limit 10;
```



```
        time         | band | rx_grid | rx_id  | tx_call | tx_grid | SNR | c2_noise | drift |   freq    |  km   | rx_az | rx_lat  | rx_lon  | tx_az | tx_dBm | tx_lat  | tx_lon  |  v_lat  |  v_lon
---------------------+------+---------+--------+---------+---------+-----+----------+-------+-----------+-------+-------+---------+---------+-------+--------+---------+---------+---------+---------
 2020-03-26 15:40:00 | 40   | KG01    | ZS3D   | ZS6GOO  | KG33xw  |  -9 |     -999 |     0 |  7.040095 |   744 |    70 |   -28.5 |      21 |   247 |     23 | -26.062 |  27.958 |   -28.5 |      21
 2020-03-26 15:40:00 | 40   | KG01pm  | ZS3RF  | ZS6GOO  | KG33xw  | -14 |     -999 |     0 |  7.040091 |   711 |    69 | -28.479 |  21.292 |   246 |     23 | -26.062 |  27.958 | -28.479 |  21.292
 2020-03-26 15:40:00 | 40   | KG01    | ZS3D   | ZS6BQQ  | KG34xb  | -12 |     -999 |    -1 |  7.040065 |   749 |    69 |   -28.5 |      21 |   246 |     27 | -25.938 |  27.958 |   -28.5 |      21
 2020-03-26 15:40:00 | 40   | KG01pm  | ZS3RF  | ZS6BQQ  | KG34xb  | -16 |     -999 |    -1 |  7.040061 |   717 |    68 | -28.479 |  21.292 |   245 |     27 | -25.938 |  27.958 | -28.479 |  21.292
 2020-03-26 15:40:00 | 40   | KG01    | ZS3D   | ZS3D/N  | KG01pn  | -12 |     -999 |    -1 |  7.040203 |    34 |    76 |   -28.5 |      21 |   256 |      0 | -28.438 |  21.292 |   -28.5 |      21
 2020-03-26 15:40:00 | 15   | JO65df  | OZ7IT  | ZS1SCI  | JF96hd  | -24 |     -999 |     0 | 21.096184 |  9924 |   175 |  55.229 |  12.292 |   356 |     33 | -33.854 |  18.625 |  55.229 |  12.292
 2020-03-26 15:40:00 | 20   | EM72go  | K4JOP  | ZS1LCD  | JF95fx  | -24 |     -999 |     1 | 14.097113 | 13121 |   114 |  32.604 | -85.458 |   292 |     30 | -34.021 |  18.458 |  39.867 | -45.445
 2020-03-26 15:40:00 | 20   | OK03gr  | HS0ZKM | ZS1LCD  | JF95fx  | -25 |     -999 |     1 | 14.097002 | 10147 |   235 |  13.729 | 100.542 |    74 |     30 | -34.021 |  18.458 |  37.092 |  29.394
 2020-03-26 15:40:00 | 20   | EL99la  | NSBSWL | ZS1LCD  | JF95fx  | -29 |     -999 |     1 | 14.096996 | 12567 |   117 |  29.021 | -81.042 |   290 |     30 | -34.021 |  18.458 |  39.043 | -34.203
 2020-03-26 15:40:00 | 10   | JN49cm  | DK6UG  | ZP9CTS  | GG22ew  | -29 |     -999 |    -4 | 28.126105 | 10589 |   233 |  49.521 |   8.208 |    36 |     27 | -27.062 | -55.625 |    58.6 |  -36.13
(10 rows)
```

The output to the screen is paged, press the space bar for the next page, or type q to end.

A where clause lets us specify fields of interest Note the use of single quotes, here around rx_id 'G3ZIL' and the band '30'. If omitted there will be an error message. Band fields are of type character, as we have 60eu and 80eu, hence the need for single quotes.

```
tutorial=>  select * from spots where rx_id = 'G3ZIL' and band = '30' order
by time desc limit 10;
```

3.1 Export query output to a file

In this interactive mode where you are connecting to the remote wsprdaemon.org server you can download the result of a query to the current directory of your local computer using the following query.

```
tutorial=>  \copy (select * from spots where rx_id = 'G3ZIL' and band =
'30' order by time desc limit 5) to 'timescale.csv' with csv;
COPY 10
```

The \ before the copy signifies an export to a file on the client computer. Note that the postgreSQL query must be within parentheses. There are limitations in the ability of the csv format to handle some aspects of postreSQL output, such as NULLs; full details are in the postgreSQL documentation[3].

The output of the query on the local computer is:

```
pi@raspberrypi:~ $ cat timescale.csv
2020-03-26 16:12:00,30,IO90hw,G3ZIL,EA5CYA,IM99,-16,-
999,0,10.140154,1274,179,50.938,-1.375,359,23,39.5,-1,50.938,-1.375
2020-03-26 16:10:00,30,IO90hw,G3ZIL,SM5FLM,JO79wj,-25,-
999,1,10.140209,1436,42,50.938,-1.375,237,23,59.396,15.875,59.396,15.875
2020-03-26 16:10:00,30,IO90hw,G3ZIL,OK2UGG,JN89do,-28,-
999,0,10.140265,1261,90,50.938,-1.375,283,13,49.604,16.292,50.938,-1.201
2020-03-26 16:10:00,30,IO90hw,G3ZIL,HB9LAG,JN47,-23,-
999,0,10.140169,842,113,50.938,-1.375,301,23,47.5,9,50.938,-1.375
2020-03-26 16:10:00,30,IO90hw,G3ZIL,EA6DT,JM19kp,-9,-
999,0,10.140223,1298,164,50.938,-1.375,347,20,39.646,2.875,50.938,-1.375
```

The time format is directly usable in Excel or other spreadsheets as date time.

Simple expressions in a query

Here is an example of a search for a particular rx_id and band that only lists SNR above a threshold of -10 dB, where only the time, tx_call and SNR are requested as output (time is there by default). As SNR is a numeric column we can use the mathematical operator >, and no single quotes around its numeric value -10. Double quotes are, however, needed around both instances of SNR as the column name contains capitals, the same is true for tx_dBm.

```
> select time, tx_call, "SNR" from spots where rx_id = 'G3ZIL' and band =
'40' and "SNR" > -10 limit 10;
```

3.2 Wildcards

Queries can include wildcards, where % matches zero or more characters or numbers. This example uses like and the wildcard symbol % for any tx_grid with characters JN for rx_id 'G3ZIL' and where band = '60', remembering the quotes:

```
> select time, tx_call, tx_grid, "SNR" from spots where tx_grid like 'JN%'
and rx_id = 'G3ZIL' and band = '60' limit 10;
```

The _ character represents one character or number. If needed there is not like to exclude.

This example shows the syntax for a query where tx_grids can be any in EN or FN and receive grids any in IO or JO and the band is 30, note the required use of parentheses around the or pairs:

```
> select time, tx_call, tx_grid, rx_id, rx_grid, "SNR" from spots where
(tx_grid like 'EN%'or tx_grid like 'FN%') and (rx_grid like 'IO%' or
rx_grid like 'JO%') and band = '30' limit 10;
```

3.3 Mathematical operations

Mathematical operations on one or more columns are allowed; column names within the mathematical expression (here "SNR" - "tx_dBm" + 30) must be in double quotes. In this example we calculate and output the SNR normalised[4] to a transmit power of 30dBm (1 watt) by subtracting tx_dBm from SNR and adding 30. The as gives the resulting column a name.

[3] See https://www.postgresql.org/docs/9.2/sql-copy.html

[4] We are, of course, aware of the pitfalls of any attempt at normalisation, power may not be correctly reported and the actual radiated power and directional characteristics of the transmit and receive antennas are not often well known.

```
select time, tx_call, tx_grid, "tx_dBm", "SNR", ("SNR" - "tx_dBm" + 30) as
"Norm_SNR" from spots where tx_grid like 'JN%' and rx_id = 'G3ZIL' and band
= '30' limit 10;
```

3.4 Simple statistics

Simple statistics can be obtained as in these examples, having done a count first to check the validity of the results. Here we have chosen a time interval to approximately span local night.

```
> select count("SNR")  from spots where band = '40' and rx_id = 'KPH' and
tx_call = 'K4APC' and time > '2020-03-25T03:00:00Z' and time < '2020-03-
25T15:00:00Z';
```

Here we use round to round the avg to the nearest integer

```
> select round(avg("SNR")) from spots where band = '40' and rx_id = 'KPH'
and tx_call = 'K4APC' and time > '2020-03-25T03:00:00Z' and time < '2020-
03-25T15:00:00Z';
```

This example is for max, we can also use min

```
> select max("SNR") from spots where band = '40' and rx_id = 'KPH' and
tx_call = 'K4APC' and time > '2020-03-25T03:00:00Z' and time < '2020-03-
25T15:00:00Z';
```

We can calculate standard deviation with stddev, and here we have used trunc with 1 to give the result to 1 decimal places. We cast the result of stddev as numeric for trunc to work.

```
> select trunc(cast (stddev("SNR") as numeric),1) from spots where band =
'40' and rx_id = 'KPH' and tx_call = 'K4APC' and time > '2020-03-
25T03:00:00Z' and time < '2020-03-25T15:00:00Z';
```

3.5 Query on the azimuth angle at the receiver.

wsprnet.org calculates the azimuth at the transmitter, tx_az, the initial bearing of the path to the receiver. However, wsprnet does not calculate the azimuth on arrival at the receiver. It is the arrival azimuth, rx_az, that is needed if one is looking to use WSPR spot information to help evaluate the directionality of receive antennas. Except for special cases, such as receiver and transmitter on the same longitude, or where the path distance is less than a few hundred km, tx_az and rx_az are not simply 180˚ apart. This example is for receiver KPH where band is '40' and we search for transmitters from azimuths between 60˚ and 90˚ where distance is between 1000 and 2000 km.

```
> select time, tx_call, tx_grid, km, rx_az, "SNR" from spots where rx_id =
'KPH' and band = '40' and km > 1000 and km < 2000 and rx_az > 60 and rx_az
< 90 limit 10;
```

3.6 Query on the vertex latitude

We have added a calculation of the latitude and longitude of the position of the vertex of the great circle path between transmitter and receiver. The vertex is the most northern or southern position. In this example we select a series of parameters where the band is '40' and the vertex latitude is > 60 (i.e. above 60˚N). Where the path is predominantly north-south the vertex is most likely to be at either the receiver or transmitter.

```
> select tx_call, rx_id, tx_lat, tx_lon, rx_lat, rx_lon, v_lat, v_lon,
"SNR" from spots where band = '40' and v_lat > 60 limit 10;
```

3.7 Query with time

A start time can be given using time >, and with a subsequent and time < to specify an interval. The time string format follows rfc3339, that is YYYY-MM-DDThh:mm:ssZ (where

Z denotes UTC, probably best to keep to queries in UTC). Note the T between date and time and the single quotes around the time string.

```
> select * from spots where  band = '160' and rx_id = 'KPH' and time >
'2020-03-25T12:00:00Z' order by time limit 10;
```

4. Single line query using psql

It is possible to connect to the wsprdaemon database, execute a postgreSQL query and return the output to a local file from a single command line. The password is set as an environment variable; psql is called with the userid, database name, host address and flags to direct output to a comma separated values file (note the comma after -F to designate the separator) in the current directory. Note that the select query is within double quotes.

```
PGPASSWORD=JTWSPR2008 psql -U wdread -d tutorial -h wsprdaemon.org -A -F, -
c "select * from spots where rx_id = 'N6GN' order by time desc limit 10" >
mydata.csv
```

5. Command line queries to the spots table using a bash script: wdquery_ts.sh

A bash script wdquery_ts.sh is available[5] as an alternative quick-look command line means of accessing receiver-centred functions.

The usage message below shows the options (as of 28 March 2020). Options are self-explanatory, rev-time is latest spot first. Invoke using ./wdquery_ts.sh if in a working directory, or as wdquery_ts.sh if you place the script in a searched path such as /usr/local/bin.

```
pi@Pi4-Dev:~ $ ./wdquery_ts_1-1.sh -h
Usage: wdquery -h this message
You must have gawk and postgreSQL installed to use this utility
       Mandatory parameters
               -b band   (One of: All, 2200, 630, 160, 80, 60, 40, 30, 20,
17, 15, 10, 6, 4, 2, 70, 23)
               -r rx_id  (ID of the receiver whose spots to list.)
       Options
               -c count  (Number of lines of spots to output.)
               -f type   (Options are csv mat. Default is csv. Mat file for
                          numeric data only.)
               -o order  (km time rev-time or band. Default is rev-time.)
               -p hours  (Hours of data to read, ending now. Default 1.)
               -u        (Unique calls only.)
```

The script can be customised by setting up your own default values using an editor to alter their values. For your most common set of options the script command ./wdquery_ts.sh will then be all that is needed. The defaults are:

```
# set up default values
#WSPRDAEMON_TIMESCALE_HOST="localhost"            # Use this if running on server
#WSPRDAEMON_TIMESCALE_HOST="wsprdaemon.org"       # Use this for remote access
WSPRDAEMON_TIMESCALE_HOST="wsprdaemon.org"        # Use this for remote access
N_LINES=5000                                      # Default number of lines
T_SPAN=1                                          # Default is one hour interval
FILE_OUT="FALSE"                                  # Default to standard output
file_type="csv"
RX_ID="KPH"                                       # RX_ID default
BAND=40                                           # BAND default
BAND_LIST="band = '${BAND}'"                      # Default single band listing
ORDER="order by time desc"                        # Default order newest first
ORDER_BY_TIME="order by time desc"
```

[5] From Gwyn Griffiths, G3ZIL, gwyn@autonomousanalytics.com

```
UNIQUE="FALSE"                                # Default is list each spot
```

To change the default band listing to All, BAND_LIST should be changed to:

```
BAND_LIST="band <> '22'"    # that is, not a number not in the band list
```

Note the use of single and double quotes.

To change the default to order by time rather than reverse time change to:

```
ORDER="order by time"
```

Change FILE_OUT or UNIQUE to "TRUE" if you would rather file output or uniques as your default. Note that the Matlab and Octave compatible .mat file is only for the numeric data, see the csv file for the numeric column order and variables. Downloaded files are saved to the /tmp directory as wdquery.csv or wdquery.mat.

6. Read access via Python

A skeleton Python script is provided in Annex B.

7. Queries from the noise table

These are of the same form as for the spots table, but where 'spots' occurs in the queries, use 'noise' instead.

Annex A. Performance metrics[6]

We are no database experts; these performance metrics are practical rather than rigorous. They refer to an Influx database of up to 10.7 million spots and a Timescale database with 18.5 million spots. During these tests the wsprdaemon databases were live - that is, up to 10,000 spots are being added each three-minute cycle. The server was also updating a handful of Grafana plots. Query times are unlikely to be shorter as spots are added.

A.1. Remote use of the command line interface as described in Section 4.

On a Raspberry Pi 4, for a query of the last 10,000 spot lines to a local csv file, when

- Influx consistently 17 s
- Timescale: ~13.2, 3.8, 3.3, 3.6, 3.5 s. Median seems to be about 3.5s. Peak coincided with a scrape from wsprnet, or an Internet slow down?

Query to extract to a csv file the unique spots over a 24 hour period.

- Timescale 8.8, 6.2, 5.9s. Median 6.2s.
  This for 5739 spots in total and finding 189 uniques.

Query to extract to a csv file the unique spots over a 7-day (168 hour) period.

- Timescale 1m55s, 1m37s, 2m4s. Median 1m55s.
  This for 45703 spots in total and finding 477 uniques.

A.2 Command line directly on the server

Using the same script on the server itself:

- Influx much the same at ~17 s
- Timescale 1.35, 1.32, 1.45 s. Median about 1.35s.

Query to extract to a csv file the unique spots over a 24 hour period, same as above 5739 spots in total and finding 189 uniques.

- Timescale 0.93, 0.94, 0.89s. Median 0.93s.

Query to extract to a csv file the unique spots over a 7-day (168 hour) period.

- Timescale 48.8, 54.5, 50.2s. Median 50.2s.
  This for 45732 spots in total and finding 477 uniques.

A.3 Using the databases in interactive mode

Influx and Timescale provide two tools to assess 'query cost': explain and explain analyze. Here we use Explain Analyze, which precedes the actual query:

```
explain analyze select * from spots where rx_id='G3ZIL' and band = '40' and
v_lat > 60 limit 100
```

Resulted in the following prediction on Influx

```
├── execution_time: 40.118612ms
    ├── planning_time: 136.961053ms
    ├── total_time: 177.079665ms
```

and on Timescale

```
Planning Time: 0.556 ms
Execution Time: 394.580 ms
```

[6] See www.outfluxdata.com/assets/20190610_Timescale_WhitePaper_Benchmarking_Influx.pdf

Annex B. Skeleton of a Python script to read the wsprdaemon Timescale database

This skeleton shows how to use the adapter psycopg2 on a local computer to read data from the wsprdaemon.org Timescale database. This needs psycopyg2 and postgreSQL to be installed (see section 0). The line select_sql can be changed to whatever query you wish to use. Note the earlier comments on the need to double quote SNR and tx_dBm if used.

```
pi@Pi4-Dev:~ $ cat ts_spot_read.py
#!/usr/bin/python
# ts_spot_read.py    Gwyn Griffiths March 2020
#  See https://wiki.postgresql.org/wiki/Psycopg2_Tutorial  for further details

import psycopg2
import sys

N_RECS=sys.argv[1]
select_sql="SELECT * from spots ORDER BY TIME DESC LIMIT " + str(N_RECS)
print(select_sql)

try:
        # connect to the PostgreSQL database
        print("about to connect")
        conn = psycopg2.connect("dbname='tutorial' user='wdread'
host='wsprdaemon.org' password='JTWSPR2008'")
        # create a new cursor
        cur = conn.cursor()
        cur.execute(select_sql)
        rows = cur.fetchall()
        for row in rows:
          print (row)
        # close communication with the database
        cur.close()
except:
        print ("Unable to connect to the database")
finally:
        if conn is not None:
            conn.close()
```