piWebCAT

1.101

G3VPX

Table of contents

1. Introduction 4
   1.1. Acknowledgements etc 5
   1.2. About this document 6
   1.3. What is piWebCAT ? 7
   1.4. Specification 9
   1.5. Dual VFO switching. 13
   1.6. Audio gain swapping 15
   1.7. Indicator controls. 'LED' option 16
   1.8 Raspberry Pi preloaded SD card 17
   1.9 Web browser choice 18
2. Using piWebCAT 19
   2.1. Getting started with piWebCAT 19
   2.2. Window size and positioning 21
   2.3. Button and slider controls 23
   2.4. Text display box 24
   2.5. Indicator controls. details. 26
   2.6. Tuning controls and RIT/XIT 27
   2.7. Station log 28
   2.8. Learning guide: Intro, hardware 31
   2.9 Learning guide: Hamlib 32
   2.10 Icom --vfo mode. Everyone read 34
   2.11 Learning guide: Transceiver-H-A 36
   2.12 Learning guide: Transceiver-H-B 39
   2.13 Learning guide: Fixed controls 41
   2.14 Learning guide: Transceiver-H-C 42
   2.15 Learning guide: Trnscvr-H-C-NV 43
   2.16 Learning guide: button groups 45
   2.17 Learning guide: mempad & more 46
   2.18 Learning guide: Hamlib data ? 47
   2.19 Using web-browser diagnostics 48
3. Database tables - configuration 51
   3.1. Introduction 51
   3.2. Configuration systems. 52
   3.3. A configuration strategy 54
   3.4. Editor page operation 55
   3.5. Database tables - overview 57
   3.6. Button and slider numbering 58
   3.7. README - setup reading 60
   3.8. README - reserved buttons 63
   3.9. README - reserved codes 65
   3.10. README - vx and abx 66
   3.11. Memories, channel switching. 67
   3.12. Frequency up/down buttons 69
   3.13. MySQL, scripts, cloning rigs 70
   3.14. **MySQL front, backups ** 72
   3.15. MySQL commands and scripts 73
   3.16 Cloning radios in piWebCAT 77
   3.17. Useful MySQL scripts 80
   3.18. Web server updates with FTP 81

3.19. Browser choice, memory leaks 82
4. ASCII text CAT - configuraton 84
4.1. Command masks on ASCII 84
4.2. Table - buttons 86
4.3. Table - catcodes 88
4.4. Table - sliders 90
4.5. Table - meter 92
4.6. FTdx101D VCT example 93
4.7. ASCII configuration - examples 94
5. Yaesu five-byte radios (FT920 etc) 99
5.1. YAESU5 introduction 99
5.2. FT847, FT817, FT818 100
5.3. FT920 101
5.4. FT920 - problems encountered 102
5.5. FT1000MkV etc 104
5.6. Example: FT818 frequency. 105
5.7. Example: FT920 frequency 108
5.8. Example: FT920 ant. switching 110
5.9. FT920 clarifier on/off 111
5.10. FT920 clarifier slider tuning 113
5.11. FT920 and FT818 rigfix 115
6. Icom CIV configuration 117
6.1. CI-V configration system 117
6.2. Table - buttonsciv 119
6.3. Table - catcodesciv 121
6.4. Table - sliderscv 122
6.5. Table- meterciv 124
6.6. CI-V configuration - examples 126
7. Common (all configurations) tables 132
7.1. Table - lookup 132
7.2. Table - rigs 133
7.3. Table - settings 134
7.4. Table - timing 135
7.5. Table - bands 137
8. HAMLIB configuration - rigctrld 138
8.1. Hamlib - introduction 138
8.2. Hamlib - radio selection 140
8.3. Hamlib - installation / update 141
8.4. rigctl, rigctld - documentation 142
8.5. rigctl - at the command line 143
8.6. rigctld - piWebCAT interface 144
8.7. rigctld - supported rig controls 145
8.8. rigctld - table buttonshl 147
8.9. rigctld - table catcodeshl 148
8.10. ridctld - slidershl 150
8.11. rigctld - meterhl 152
8.12. rigctld - text box data items 153
8.13. rigctld - mode and bandwidth * 154
8.14. rigctld - unsupported comnds. 155
8.15. \send_cmd_rx - SSB mod source 156
8.16. \send_cmd_rx - contour & APF 157
8.17. \send_cmd_rx IC7000 duplex 160

8.18. Generating a Hamlib error log 161
9. S meter and Tx meters 162
9.1. Meter background images 162
9.2. Meter calibration 163
10. VOIP for Rx and Tx audio - Mumble 164
10.1 Introduction, security, comms. 164
10.2 Structure and configuration 165
10.3 mumble-client RPi 166
10.4 mumble client - Windows laptop 167
10.5 Using mumble - some notes 168
11. How it works 169
11.1. SD card features, website folder 169
11.2. Database access 171
11.3. piWebCAT startup 172
11.4. piWebCAT request queue 175
11.5. MOX and Smeter/ Tx meter 176
11.6. Button operation 177
11.7. Slider operation 178
11.8. Frequency control - tuning 179
11.9. Band switching 181
12. piWebCAT code 183
12.1. Basic information 183
12.2. HTML code 184
12.3. PHP code 185
12.4. PHP example - serial init. 187
12.5. Javascript code and jQuery 189
12.6. MYSQL data client, server 190
12.7. Ajax, MYSQL sliders example 191
12.8. Data transfers after startup 196
13. The piCAT RS232 and CI-V PCB 197
13.1. Hardware: RS232, G3VPX PCB 197
13.2. PCB schematic 199
13.3. PCB component list 200
14. Downloads, support etc 201
14.1. File downloads 201
14.2. Support. Supply SD card & PCB 202
14.3. Develpment tools 203
14.4. Configuring SD card 205
14.5. installing Hamlib on the SD card 211
14.6 Installing Mumble on the SD card 212
14.7 Installing Mumble on Windows 218
14.8 Installing Mumble on Android / IOS 222
14.9. Javascript debug popup 223

1.1 Acknowledgements

Acknowledgements

I Ian Sumner, G3VPX recognise the various organisations that have products that have contributed
to the development of piWebCAT and their copyright to these products.

These include, but are not limited to:

The jQuery Foundation for jQuery code and addon: jquery.mousewheel.js.
Dave Furfero and Manuel Gumpinger for jquery-ui-touch-punch.js.
MariaDB Foundation for MariaDB Database Software
Kayson Group Inc for phpGrid software for which I hold a purchased 'Ultimate' license for OEM distribution.
(The phpGrid downloads in include jQuery and jqGrid which are also available as free internet downloads)
IBE-Software for the HelpNDoc document authoring system.
Microsoft for their Expression 4 web authoring system (free download)
The Raspberry Pi foundation for their Raspberry Pi OS (previously Raspbian) and bundled software.
Hamlib: The Ham Radio Control Library–Hamlib. (hamlib.github.io)
Mumble: An open source, low latency, high quality voice chat application.
Rémy Sanchez and Rizwan Kassim for PhpSerial.php serial port code.
Tan Li Hau for alternative javascript JSON parser code (modified).. that doesn't leak memory.
Ansgar Becker for HeidiSQL
Nils Hoyer for MySQL Front

MySQL / MariaDB

MariaDB Server is one of the most popular database servers in the world.
It is made by the original developers of MySQL and guaranteed to stay open source.
Notable users include Wikipedia, WordPress.com and Google.
Is used by the Raspberry Pi Foundation.

Most references in this document refer to MariaDB as MySQL.

Hamlib

The Ham Radio Control Library–Hamlib, is a project to provide programs with a consistent Application Programming Interface (API) for controlling the myriad of radios and rotators available to amateur radio and communications users.
In piWebCAT, it translates a common, standard command set into the correct CAT commands for a selected radio.

1.2 piWebCAT - a Raspberry Pi web server for CAT control of radios

This document was generated with **HelpNDoc** help authoring software:

- As a 222 page printable PDF document
- As an indexed website on http://piwebcat.g3vpx.net
- As an indexed website on the piWebCAT server and accessed from the **help** button.

The document is in six sections:

- **Introduction and specifications**

- **A Getting started and learning section**
  This includes a Learning Guide which is based on a progression of three generic transceiver control configurations. This uses the Hamlib rigctl API which supports 250 radios.

- **A large configuration reference section.**
  (piWebCAT is highly user-configurable with configuration information stored in a MariaDB (MYSQL) database on the Raspberry Pi web server.)

- **A section on the use of Mumble for Voice Over IP transmission of Rx and Tx audio.**

- **A section entitled 'How it works!'**

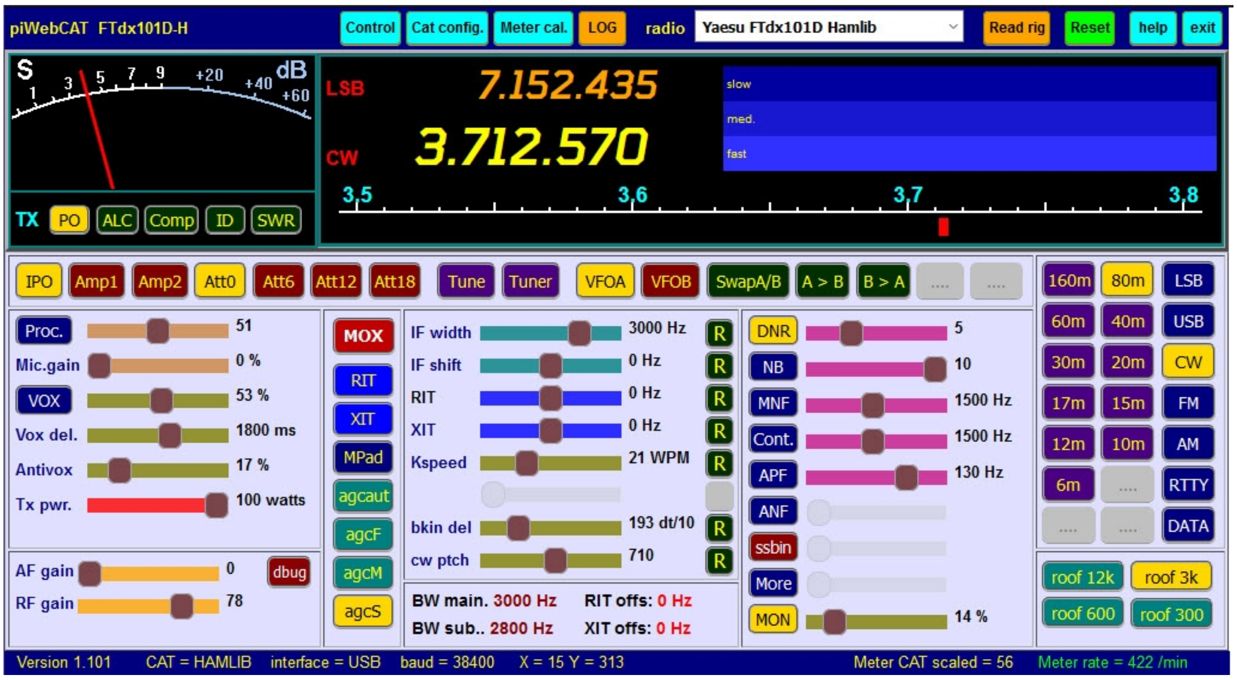
- **A section on some detailed aspects of website programming**
  - with special reference to extracting data arrays from the server database to the client (web browser) using jQuery Ajax.

- **A section containing:**
  - G3VPX piWebCAT PCB
  - File downloads
  - Support (iogroup)

- **Development tools**
  Discusses two tools without which I could not have done this development:
  - A versatile storage oscilloscope with ASCII and HEX decode of serial data.
  - The 'Inspect element' feature of Firefox developer (and other web browsers).

1.3 piWebCAT - CAT control from a web-browser

I have recently developed **piWebCAT** (January 2021).
piWebCAT provides CAT control from a web browser.
It uses a **Raspberry Pi 4B** computer with Apache web server, MYSQL database and PHP7 server programming.
It works well on PC based browsers.
It is fine on my fast modern Android 7 tablet. (Touch tuning a little slower on an older tablet)



radio Yaesu FTdx101D Hamlib

LSB 7.152.435
CW 3.712.570

Proc. 51
Mic.gain 0 %
VOX 53 %
Vox del. 1800 ms
Antivox 17 %
Tx pwr. 100 watts
AF gain 0
RF gain 78

IF width 3000 Hz
IF shift 0 Hz
RIT 0 Hz
XIT 0 Hz
Kspeed 21 WPM
bkin del 193 dt/10
cw ptch 710

BW main. 3000 Hz RIT offs: 0 Hz
BW sub.. 2800 Hz XIT offs: 0 Hz

DNR 5
NB 10
MNF 1500 Hz
Cont. 1500 Hz
APF 130 Hz
ANF
MON 14 %

Version 1.101 CAT = HAMLIB interface = USB baud = 38400 X = 15 Y = 313 Meter CAT scaled = 56 Meter rate = 422 /min

Above **is piWebCAT's control window.**

The buttons and sliders are highly user configurable (stored in a MySQL database on the RPi)

The other windows are **database editor** and **meter calibration**.

.

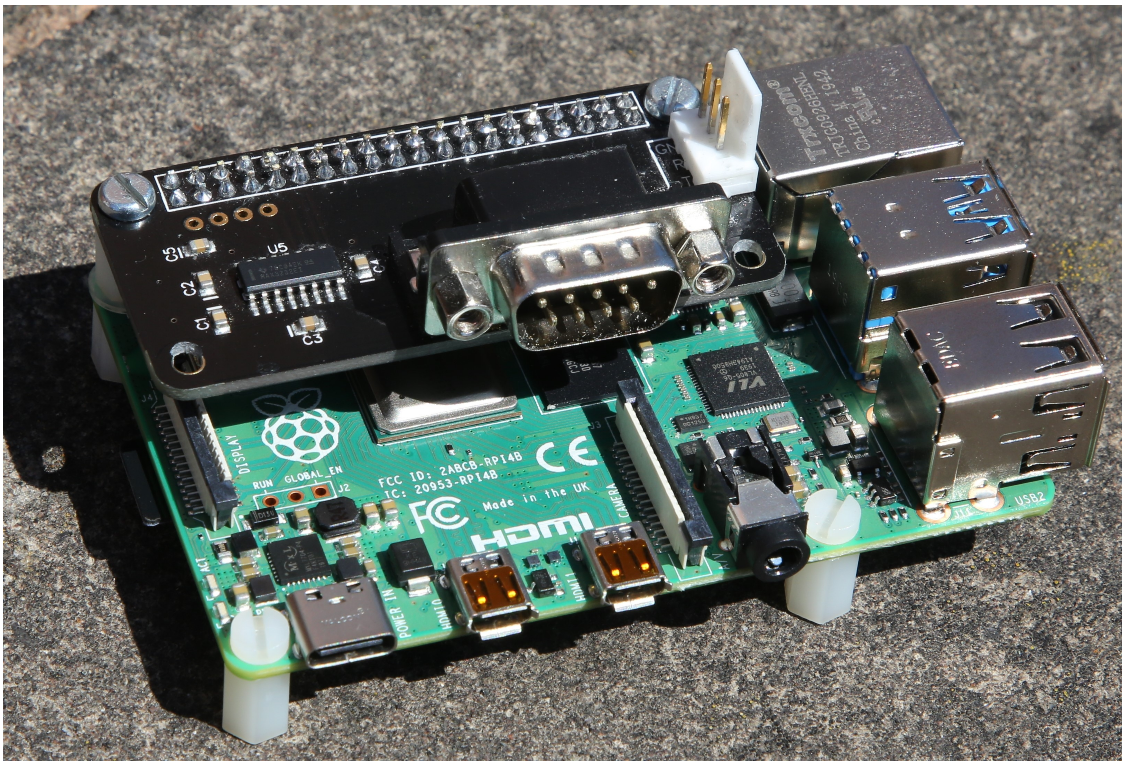
Tuning

Rapid approximate positioning in the band is by clicking (or touching) the tuning scale.

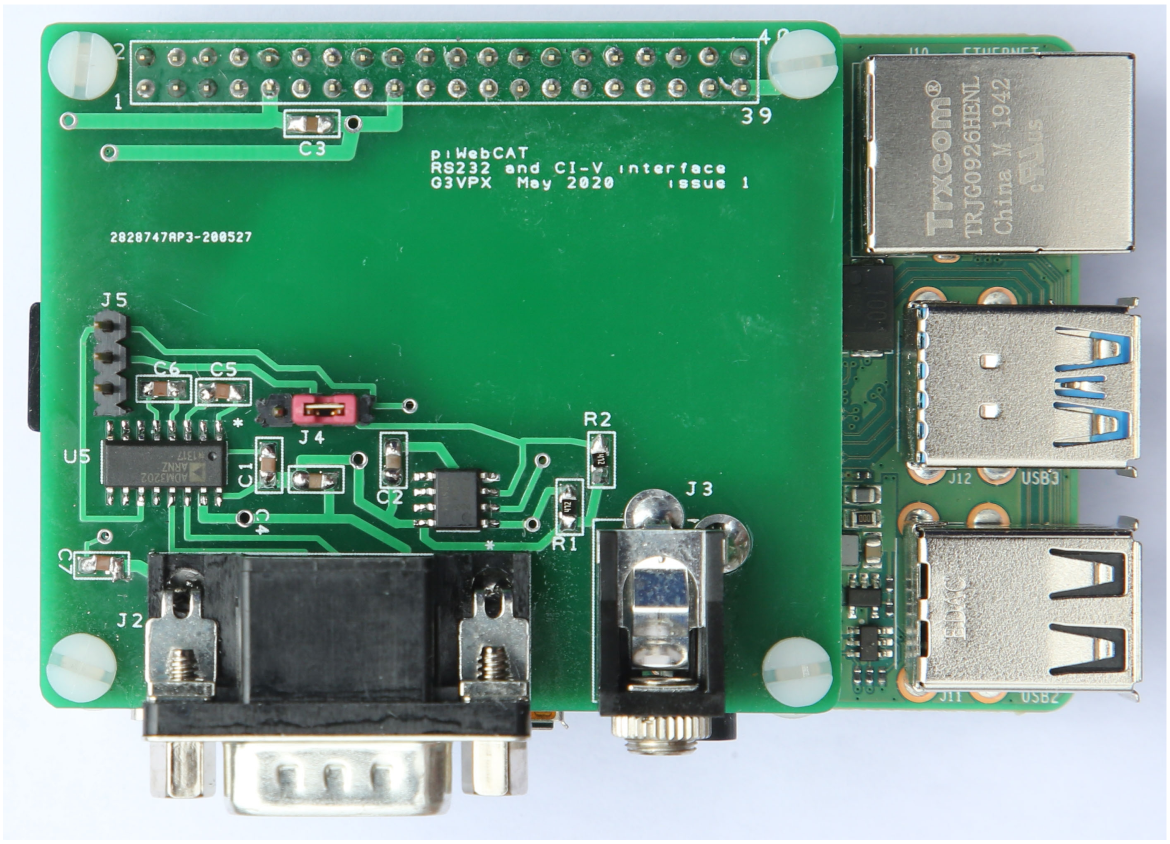
Tuning is the done in the blue tuning panel. The tuning panel has fast, medium and slow lanes.
Tuning is performed:

- by horizontal mouse pointer (or finger) dragging. (Tuning rate fast, medium or slow according to lane)
- by mouse wheel. (Tuning rate fast, medium or slow according to which lane the mouse pointer is in)

piWebCAT hardware



The piWebCAT hardware: A RPi 4B computer + piZero RS232 card + preconfigured micro SD card.



My replacement interface piCAT design has RS232 and CI-V connectors.
The PCB is full width to bring the connectors to the side of a box.
The free space on the pcb has no underlying ground plane. It could be used for development 'breadboarding'.

If you can connect to the radio via an RPi USB port, then you not need these add-on interface cards.

1.4 piWebCAT specification

Configuration options:

Using user entered CAT commands:

ASCII	ASCII text character configuration system used for Yaesu, Kenwood, Elekraft
YAESU5	for the earlier Yaesu radios using a bcd/hexadecimal command structure (FT847, FT818, F920, FT1000, MkVFT1000MP etcv)
CIV	For Icom CI-V control

Using the Hamlib rigctld AP:

- **HAMLIB** **rigctld** translates a common set of commands into CAT commands for your selected radio.
  There are 250 supported radios in the Hamlib database. Each is simply selected by number.
  eg: If FTdx101D (#10400) is selected,
  then Hamlib: **set_freq VFOA 3744000** will translate to Yaesu: **FA003744000;**

Extensive, flexible user configurability.

Configuration stored on RPi server in a MySQL database.
The database can store configurations for multiple radios.

Hardware:

Raspberry Pi 4B computer only ..... if using USB connection to radio.
+ RS232 or G3VPX RS232/CI-V GPIO card for serial data connection to radio.

Software

Preconfigured Raspberry Pi system on **micro SD card**.
Transceiver configuration databases supplied for Modern Yaesu, Yaesu FT920, Icom IC7000.
Hamlib configurations for FTdx101D, IC7000 and a generic Hamlib transceiver.

Documentation and help

This web site is also built in to piWebCAT (help button) It is downloadable as a 222 page PDF file.
This includes an 11 page learning guide based a evolving sequence of three configurations.

Frequency control:

VFOA / VFOB, swap, B to A, split (or whatever else you want to configure.)

Tuning:

A dedicated tuning scale for each band with marker. Touch or click the scale for coarse positioning.
A tuning window with thee horizontal bands: fast, medium and slow.
Tune by mouse (or finger) drag along these bands
OR three mouse wheel tuning rates according to which band the mouse pointer is in.
(Tuning rates are user definable and stored in the database)
On an android tablet, tune by finger drag or by mouse wheel on bluetooth or OTG mouse.
User configurable Up and Down buttons, eg: -12.5kHz + 12.5kHz -25kHz +25kHz.

Memories.

A button can be configured to popup a memory selector numeric keypad to select a memory by number'
On clicking OK, the selected memory is sent in a 'memory to VFO' CAT command..

Bands:

160m, 80m, 60m, 40m, 30m, 20m, 17m, 15m, 12m, 10m, 6m, 4m, 2m and 70cm.

Tuning by UP and DOWN buttons for **VHF/UHF channels**, eg: +12.5kHz and -12.5kHz
This is one of a small number of fixed coded functions -
but you retain control of frequency shift, button choice and captions and colour.
Slider and button settings for latest band on each VFO are remembered for fast VFO switching.
(same band or cross band)

27 slider controls

- some predefined + some spare for user allocation ... ***but you can redefine them all.***
Each slider has an adjacent value display (with scaling, dec.point, optional table lookup and units)
Sliders are in three groups:

- COMP, Mic Gain, Vox gain, RFpower, AF gain and squelch (probably unchanged) + 2 spare.
- Nine sliders with a reset to default button - default defined by you in the database -
- Nine sliders with an adjacent button: mainly on/off.
  - the association of button and slider is database defined and optional.

Using web browser Firefox, Sliders can be slowly moved with the mousewheel (allowing effective RIT tuning)

Sliders as indicators - Selected sliders can be optionally enabled for frequent periodic update of their marker position and of their adjacent numeric parameter value.

66 button controls + 24 extra buttons on a popup window.

A few buttons need to be of fixed function: ie: Band buttons and VFOA and VFOB.
Some are associated with sliders (eg: DNR on/off - DNR level.) but you can change what they all do.
Five are dedicated TX meter function selectors - but you choose which five Tx meters to offer.
The rest are under user configuration control for:
- the CAT commands that they generate.
- their action: S = single momentary, T = toggling, G = grouped (have common code) etc
- their captions and the background colour of the button.

Buttons as indicators - Selected buttons can be enabled for frequent periodic status update.
These can optionally function as 'LED' indicator lamps:
- In their OFF state,they are black with dark red text.
- In their ON state they light up in a bright colour of your choice with black text.

Audio switching on dual receiver transceivers

Option on VFO swapping to automatically mute background receiver and switch the foreground receiver to its last audio level. *(needs CAT access to background VFO see section 2.10 Icom --vfo mode)*

Default settings of controls:

At start up, frequency and mode and settings for buttons and sliders are loaded from the radio.
Thereafter, frequency, mode and MOX update on piWebCAT, if changed on the radio.

In addition, every button and slider control has the option to set **active=S** = *sync* (instead of just Y = yes).
All such **sync** controls are then continually polled in a circular list and updated from their current rig setting.
Sync status thus is restricted by user choice to selected controls to minimise the data bandwidth.
The polling interval for the list is user configurable.
Any such control's update interval = polling interval / total no of sync controls.

A 'Reconnect' button is therefore provided to reload the settings.
However, this is only needed if changes are made on the radio:
If you use piWebCAT to change band or VFO on the radio, the radio responds immediately and piWebCAT's controls are updated automatically within 5 seconds.

However, piWebCAT remembers the most recent settings for each VFO and so this 5 second delay does not occur on repeated swapping between VFO A and VFO B.
(N*eeds CAT access to background VFO see section 2.10 Icom --vfo mode)*

S meter / Tx meters

On receive, the meter this is an S meter with an appearance similar to the FTdx101D.
On transmit, the meter displays one of five button selectable options.
The Tx values for display and their CAT controls are defined in the database for your radio.
I provide five Tx meter backgrounds for Power, ALC, Compression, PA ID and SWR.
These are simple 250 x110 bitmap images ... you can change them (please keep the arc !!)
Accuracy of display is achieved for each of the six installed metering functions by the provision of 20 point interpolated calibration tables in the database (and an easy calibration procedure)

Database editing.

piWebCAT has a spreadsheet-like editor grid for each of the database tables.
Editing is directly onto the grid. Only records for the selected radio are presented.
The table can be exported to a CSV file for import into Excel. They can then be printed.
The database is standard MySQL (MariaDB) and so can be edited by other database editor/toolkits.
For most of the development, I used the free **MySQL Front.** This can export radio configuration and so facilitates the exchange of configuration data between users. (See section 14.1 Downloads)
More recently, I have used **HeidiSQL**: another excellent, free database editor / toolkit.

Rig duplication - SQL scripts

I provide SQL scripts which include duplicating a whole rig configuration (with a different name!).
This allows you to preserve my existing configurations whilst experimenting on a copy.

Design philosophy

I have attempted to maximise the use of a user editable database in configuring piWebCAT.
This means that there are very few controls tailored to the features of particular radios.
In the direct ASCII, CI-V and YAESU5 configurations (ie: Not using Hamlib), a small minority of commands cannot easily be supported. A example of this is **Rx clarifier on the Fdx101D.**
If we compare the FTdx101D **IF shift** and **clarifier** commands.

- **IF shift is not a problem**. A single read/write command is used, eg IS00-0340; for -340Hz (current VFO.)
  piWebCAT can display a centre-zero slider with text -340Hz and a centre zeroing reset button.
- **Rx Clarifier is difficult.** The FTdx101D has an on/off command and a clear command.
  There are separate write-only commands for shift up increment and shift down decrement.
  I combined these into a single control for the FT920 by using a 'rigfix' (See below).
  I can do it for other radios if requested. (But the use of a Hamlib configuration is probably a better solution)

Amateur radio logging system *For auto-time entry needs internet or RPi real time clock (£5)*

The log is stored in a table in the RPi MySQL database. It has the following features:

- One single button click enters: date, start time, frequency, mode, power and optional contest number.
- Date picker and time sliders are provided for 'manual' data entry.
- Optional multiple lines of text with word wrap in the remarks column (automatically expands the row).
- Searchable label column with three specific label options to highlight the row in a background colour.
- Search button.
- QRZ.com button for selected callsign.
- Export to CSV button (will launch in Excel for printing etc)
- Backup and restore of log using third party MySQL editor (eg: MySQL Front ...free download)
- Logging system can be be run separate from piWebCAT (*No auto-insertion of freq, mode nor power*)

I have provided:

- An SD card image download for the Raspberry Pi 3 or 4.
- Extensive documentation in a website.
- FTdx101D, FT847, FT818, FT920 and IC7000 database configurations as examples and as templates.
- In addition: two sets of three progressively developing learning configurations with a learning guide.

Software

piWebCAT was built using **Microsoft's Expression 4** - a free downloadable web development package.
The design was done 100% in code. This kind of web page is too complex for WYSIWYG graphical layout.

You can use an FTP client program (eg: FileZilla) to download the complete piWebCAT code from the usual Linux /var/www/html folder on the RPi. You can then edit it and play with it using Expression 4.
You need to keep a copy of the original. Javascript and PHP code can be killed by a single wrong character which can be very difficult to locate across multiple coded sections!!

The piWebCAT data path is:

Javascript in the web browser < LAN > PHP code in RPi webserver < serial OR USB > FTdx101D.

Note that the client javascript makes extensive use of **jQuery** (open source).
Communication with the server is by jQuery **Ajax** commands.
The editing grids use **phpGrid** - a purchased Chinese product that is licensed to me for OEM distribution.
phpGrid makes extensive use of javascript **jqGrid** which is open source.

RIgfix

If there are a few commands that cannot be implemented on certain radios, then a rigfix might be needed.
A rigfix is hard coded in PHP server code. A rigfix is applied to a radio by the **rigfix** field in the rigs table.
It has drop down selector data entry to select from the currently available options.

At the time of writing, the available options are FT818 and FT920 See: FT920 and FT818 rigfix

Example: Data is read from the FT920 in large blocks spanning multiple controls. piWebCAT reads for each control separately. The rigfix provides a caching system to reduce data bandwith requirements by avoiding repetitive multiple reads of the same large data block.

Mumble

Mumble is a free, open source. low latency, high quality voice chat application using Voice Over IP (VOIP).
It is not part of piWebCAT. Its installation adds the transmission of Tx and Rx audio to piWebCAT which allows operation from a location remote from the transceiver using the combination of CAT control and audio.
It is one of a number of VOIP applications that could operate in parallel with piWebCAT.
It appeared to be the most suitable package for this purpose. I have used it in QSOs with piWebCAT.
It does not carry the burden of video transmisison which is a feature of some of the alternatives.

1.5 Dual VFO switching

This doesn't work with Icom rigs and other rigs with no access to background VFO settings....

****** you must read Learning guide 2.10 Icom --vfo mode ******

Storing settings for rapid VFO switching

Most modern radios have two 'VFOs'.
These are implemented in different ways:
eg: On my two radios:

- **IC7000** This has a single receiver and two VFO settings.

- **FTdx101D** This has two completely separate ide ntical receivers.
  Several receiver parameters are duplicated., ie: each receiver has its own separate controls.
  These include DNR. contour, notch, width, shift, RF gains, squelch, .AFgain.
  These settings will change on band switching.

The two radios above reflect the two extremes of dual VFO systems.
The FTdx101D is the more complex to deal with.

For receiver parameters, the FTdx101D CAT manual shows separate commands controlling each receiver.
eg: RL0; requests a read of noise reduction level on the main receiver and RL1; on the sub.
Thus CAT commands do not act on the ***current*** receiver but must be directed to main or sub (A or B).
piWebCAT's controls act on the the current 'VFO' : A or B.
piWebCAT directs commands to the correct receiver according to current selection.

I examined the CI-V command set of the Icom's current IC-7610 dual receiver transceiver.
I cannot find receiver specific commands. All commands appear to work on the current receiver?

piWebCAT has one set of controls, a majority which are applied to the currently selected 'VFO' or receiver.
For the dual receiver FTdx101D, we have to load the settings for the 'new' receiver (VFO) on switching to it.
(The settings will differ between bands, but because there are two separate receivers, they can differ
on the same band.)
Loading all the settings for buttons (66 + 24 on the extra buttons popup) and sliders (27) takes about 4 seconds.
This delay is a consequence of the more complex data path on a web-server based system.
To compensate for this 'speed problem', piWebCAT stores the latest settings for each VFO:

- On loading from the radio, slider and buttons settings for each VFO applied to the controls and also stored.
- Once both VFOs have been selected once, swapping between them uses the stored settings and so is rapid. (this works using cross-band or the same band)
- Any button or slider actions update the stored data.
- If a VFO is moved to a different band, then a reload occurs.

For settings that have separate values per VFO, set **vx = 'V'** in sliders, buttons, slidersicv, buttonsciv.
For setting that do not change with VFO change, set **vx = 'X'**

Important:
The DNR setting on the **IC-7000** is a single setting for both VFOs.
So if I change VFO (via CI-V or on the radio), the DNR level does not change;
If I activate the above described VFO specific storage for DNR level, then I create a problem:
eg: DNR = 4 on VFO A. I change to VFO B and set DNR = 9.
When I switch back to VFO A, piWebCAT restores its stored value of 4, but DNR on the radio stays at 9.
We prevent this by setting vx = 'X' in the the **slidersciv** record. (rather than vx = 'V'.. which will store on change)

Please see section 2.10 Icom --vfo mode

RF power level - band specific?

On initial load (for each VFO selection), the states of all controls are loaded from the radio.
Only the VFO A/B (receiver A/B) controls are stored for subsequent rapid deployment on VFO switching.
The FTdx101D does not store band-specific power levels but some radios may do this.
piWebCAT therefore reads RF power level from the radios at all VFO and band changes.

1.6 Audio gain swapping

This doesn't work with Icom rigs and other rigs with no access to background VFO settings....
****** you must read Learning guide 2.10 Icom --vfo mode ******

The **IC7000** has only one audio output channel - from the current VFO.

The **FTdx101D** has two completely separate receivers with separate audio gain controls.
On switching receivers, there appears to be no way of automatically muting the background receiver.
(Unless I have missed something in the user manual!)

piWebCAT has an option to mute the background receiver and set the foreground receiver to its latest level,
The Ftdx101D has a toggling **mute** function in the CAT menu. I do not use it because other radios don't have it.
The **rigs** table has an **afswap** field which is set to **Y** to enable the facility.
piWebCAT stores the latest foreground audio gain level for each receiver.
On VFO / receiver switching, the audio gain in the background receiver is set to zero
and the audio gain in the foreground receiver is set to its stored value.

Configuration of the AF gain slider:
One record in the **buttons** table; two records in **catcodes** ie: A and B.
The linking **code** field must be '**AFGN**' for the above described system to operate.

Using Mumble VOIP - audio source - considerations for audio gain swapping

Sections 10.1 to 10.5 describe the use of **Mumble** VOIP (Voice Over IP) for remote operation.
In a dual receiver transceiver, if both receivers are used, the remote control of audio gain is desirable.
Furthermore, the use of a received audio signal that is controlled by the audio gain control is needed
for audio gain swapping to operate.

My FTdx101D has three possible sources of receiver audio to feed into the USB audio adapter.

- **Rear panel: The 6-pin data connector**. The appears to have output only from the Main receiver.
  It has a fixed level. *The front panel and CAT audio gain controls have no effect.*
- **Read panel: A 3.5mm stereo jack** socket with Main and Sub receiver outputs separate on
  the two pins . It has a fixed level. *The front panel and CAT audio controls have no effect*.
- **The front panel headphone socket**. This combines the output from both receivers.
  The front panel audio gain controls and CAT audio gain controls are effective.

Therefore, if you want to use Mumble remotely, with access to both receivers and have separate control
of AF gain, then you have to use the headphone socket to feed the USB adapter.
In some radios, an attenuator might be needed.
In addition, piWebCAT's audio gain swapping feature (on VFO swap) can only work using CAT
audio gain control and so would have to use the headphone socket.

1.7 Indicator - controls - buttons and sliders. 'LED' button option

piWebCAT offers 27 slider controls and 66 + buttons + 24 extra buttons on a popup window.

Active controls are synchronised to their corresponding radio parameters at start up, band change
and by the Reconnect buttons. If a large number of controls are used, this update can take four seconds
(eg: with my FTdx101D configuration) The exception to this delay is a special provision on Dual VFO switching

To regularly synchronise all these controls to their radio parameters would create a large amount of CAT
traffic which would interfere with the CAT 'bandwidth' available for tuning and meter update.
piWebCAT therefore provides a means for you to configure a 'limited list' of controls for regular synchronisation.

Buttons

A button configured with **active = L** (led) will behave as a *read-only indicator light*.
It is regularly synchronised to its radio parameter at an interval configured in the **timings** table.
In its OFF state, it has a black background with red text.
In its ON state, it lights up in a bright colour of your choice with black text.
Thus the configured colour specifies the bright ON colour rather than its OFF colour.
Clicking it has no action.

A button configured with **active = S** (sync) appears and behaves like a standard button but is regularly
synchronised to its radio parameter at an interval specified in the **timings** table.

Just as all other buttons, **L** and **S** buttons are set to reflect their status on the radio at start up, at band change
and by the Reconnect button.

Sliders

Sliders can also optionally be configured with active = L or active = S.
For both these options, the control is regularly updated as descibed for buttons above.
For active = L, the slider has a different appearance. See slider sync

Configuration of update timing is described in sync timing

1.8 piWebCAT installation on a micro SD card.

I have three options:

- Supply a **pre-configured 16 Gbyte micro SD card.** This my current option.
- Supply a **zipped SD card image on DVD**. I can do this.
- Provide a **zipped downloadable SD card image.** This is over 3 Gbytes and would require extension of my web hosting contract. I would do this if demand for SD cards was unmanageable!!!

The SD card has preinstalled software as follows:

- The latest **Raspberry Pi operating system** (February 2021)
- **Apache2 websrver**
- **PHP 7** server programming language and PHP database links.
- **MariaDB** (MySQL) database system. Phpmyadmin web browser database tool.
- **Pure-ftpd** FTP server for upload / download of code etc to / from the RPi.
- **Hamlib rigctl / rigctld**. This has been developed over 12 years to support 250 radios.
  It allow piWebCAT (and other CAT systems) to use a generic command set which it translates into the commands for the selected radio.
- **Mumble-server** and **Mumble** client. This is a free, ready made VOIP (Voice Over IP) system which transmits microphone and speaker audio over LAN and so can facilitates remote operation using piWebCAT.

Preinstalled data and code

- The piWebCAT webserver. This includes HTML web page rendering, javascipt / jQuery web browser program code and PHP server code.
- **phpGrid** - this is an extensive purchased product from China. It makes use of the open source jqGrid system to provide an easy means of coding speadsheet-like grids on a web-browser.
  It is used to generate the editor grids for the configuration database.
  I have a licence for phpGrid which allows OEM distribution.
- The whole of this website linked to piWebCAT's HELP button.
- Some SQL scripts which include a script for rig duplication - section 3.13 MySQL Front

Building an SD card

This is described in 14.4 Configuring SD card, 14.5 Hamlib installation, 14.6 Mumble installation.

The process includes serial port redirection and database and FTP server setup.
These are somewhat complex and time consuming.
Their full documentation gives a useful insight into the systems and a reference guide for making changes.
It also helped me to rebuild a card for issue .... and thereby validated the instructions that I had recorded!

Low cost micro SD card duplicator

I found thiis link to be very useful: https://ccse.kennesaw.edu/outreach/raspberrypi/duplicate_sd-pc.php

I use a **Raspberry Pi 400** (keyboard with RPi4 built in), a **10 port USB3 hub** (amazon £30) and ten small **USB3 card readers** (£6 each). The **pishrink** program reduces the source card image to 8.4Gbyte.
10 copies are produced by the **dcfldd** application in 18 minutes.
(I use fast class10 cards ... for fast copying and fast piWebCAT)
They auto-expand to fill the 16Gbyte card in a few seconds on first use.

1.9 Web browser choice - some issues

Wikipedia has a good comparison of web browsers at:
https://en.wikipedia.org/wiki/Comparison_of_web_browsers

This lists sixty four web browsers. Twenty four of these are marked as discontinued.

The underlying technology is stated in the column: **Current layout engine**.
It is clear that most make use of one of four 'engines', ie: Trident, Bink, Gecko or Webkit.

I have extensively used **Mozilla Firefox** and **Firefox Developer** which use the **Gecko** engine.

Please be aware of two features of these Mozilla Firefox browsers when used with piWebCAT:

- piWebCAT's slider controls are not ideal for very fine adjustment.
  However, with Firefox, if you click the mouse thumbwheel over the slider tab, you can then use the thumbwheel for fine adjustment of the slider position. ***This is excellent for RIT tuning etc***
- **Firefox** has a problem with **memory leakage**. (Approx 50 Mb / hour compared with 1 Mb in Chrome.)
  The problem arises because piWebCAT is running around 20 background web transactions per second.
  These transactions are S meter reads and synchronisation reads for selected controls.
  (*By comparison, many web-based applications sit there doing nothing until you hit a key!!)*
  After just over 1.5 hours operation, speed has dropped to a level which is still useable but the screen image starts to break up. (Raspberry Pi 4B. Fast PC with i7 processor)
  However, a click of the reset button resets piWebCAT (5 second restart delay)
- With **Google Chrome** at three hours, the browser is still working ok, albeit with about 45% speed.
  Again - a click of the reset button resets the browser.

Memory leaks and associated problems are explained in some detail in section 3.17: Browser choice - memory leaks
Memory leaks are certainly not the sole cause of the slowing.

The issues are well documented on the internet, including the Firefox issue.

I have gone through the software design very carefully, applying all available advice to minimise memory leakage.
I am not alone in failing to completely eliminate it.
The underlying problem is piWebCAT's unrelenting background rate of client server transactions.

Note that the reset button does not change anything on the rig.
The 5 second delay is mainly due to piWebCAT's controls re copying their corresponding rig settings.

2.1 Getting started with piWebCAT

support group at **piwebcat@goups.io**

Note the Learning Guide sections 2.8 - 2.15. These use training configurations on the SD card.
They use the Hamlib database of 250 radios and so are not rig-specific.
They evolve from A to B to C with progressive addition of features.
Two versions are provided: The Transceiver-H-A-NV, ...H-A_NV and ..H-C-NV configurations do not use Hamlib --vfo mode. This is better for Icom radios. See section 2.10 - Icom issues
Please note that piWebCAT was designed on my four radios, FTdx101D, FT920, F847, IC7000.
Hamlib supports 250 radios and a existing configuration can be made to work with another radio.
This still involves experimenting: Hamlib provides a generic command syntax and uses common parameter names, eg: ROOFINGFILTER .... but not all rigs have CAT control of roofing filters!
Some features are untested. For example, I haven't run piWebCAT via USB with an Icom transceiver.
(The covid lockdown gave me the time to develop piWebCAT, but prevented me from visiting other radio hams to test is on their radios!!)

Hardware

piWebCAT was developed on Raspberry Pi 4 computers. I have not tested on RPi 3 computers.
A complete preconfigured system is available from G3VPX on a 16Gbyte Micro SD card.
A fast Class 10 card is recommended. The cost of these is now under £5.

Please see Section 14.2 Support. Supply SD card and PCB

Interface to the transceiver is one of:

- **USB** - No extra hardware required.
- **RS232** - Serial Pi Zero PCB, Serial Pi Plus PCB or a piWebCAT PCB from G3VPX (bare board)
- **Icom CI- V** - 3.5 mm mono jack - Rx and Tx data share a single wire
  - Needs a piWebCAT PCB from G3VPX.

For Rx and Tx audio support via Mumble, a **USB audio adapter** is needed (£4.50 from Pi HUT)

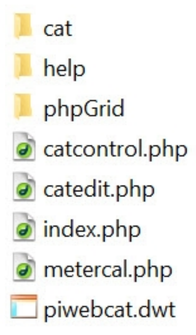
RPi user interface

piWebCAT and the Mumble audio system run on the RPi without any need for user intervention at start up.
Thus keyboard and mouse are not needed. The RPi, once configured can be treated as a 'black box'.
If keyboard mouse access is needed, then the most convenient approach is to use VNC client on a PC.
The RPi configuration includes VNC server for this purpose.

Running piWebCAT

piWebCAT is run by simply inserting the address (or URL) of the RPi in the URL bar of a web browser.
eg: 192.168.1.117.
(The Apache2 web server provides access on port 80 (ie: 192.168.1.117:80) which is the standard port for web browsing. Many website then switch to secure (https) access ... which is not needed here)

The piWebCAT 'website' on the RPi webserver.



cat
help
phpGrid
catcontrol.php
catedit.php
index.php
metercal.php
piwebcat.dwt

The 'website' structure is shown left
These files and folders are located in the RPi **web root** at **/var/www/html.**

*The **cat** folder and the **five files** are the code that I have generated.*
They are what I would normally replace in a code update.

The **help** folder contains a copy of this website and as such is accessible from the Help buttons in piWebCAT.

The phpGrid folder contains 21.4 Mbytes of phpCode for which I purchased a licence for OEM distribution.

FTP access

The RPi is configured with **pure-ftpd**.
This is an FTP server that allows access to **/var/ww/html** (the **web root)** and to the **/home/pi** folder.
Files can be uploaded and downloaded to a PC using an FTP client program such as FileZilla.

Access parameters are: IP address, eg: **192.168.1.117** password = **feline.**
User = **upload** for **web root** and **piuser** for **/home/pi.**

Modifying piWebCAT code

Microsoft's Expression 4 was used to develop piWebCAT. This is a free web development application.
The development was totally based on writing code and then testing. The visual facilities were not used.
Expression 4's site settings are configured with the same FTP access parameters as listed above.

The website files and folders are downloadable as a zip file (see section 14.1).
They can be unzipped into a folder of your choice on your PC and then that folder is specified as the web root for Expression 4 development.

MariaDB (MySQL) database.

piWebCATs; configuration data is stored in a MySQL database named**: radios**.

The data can be edited in two ways:

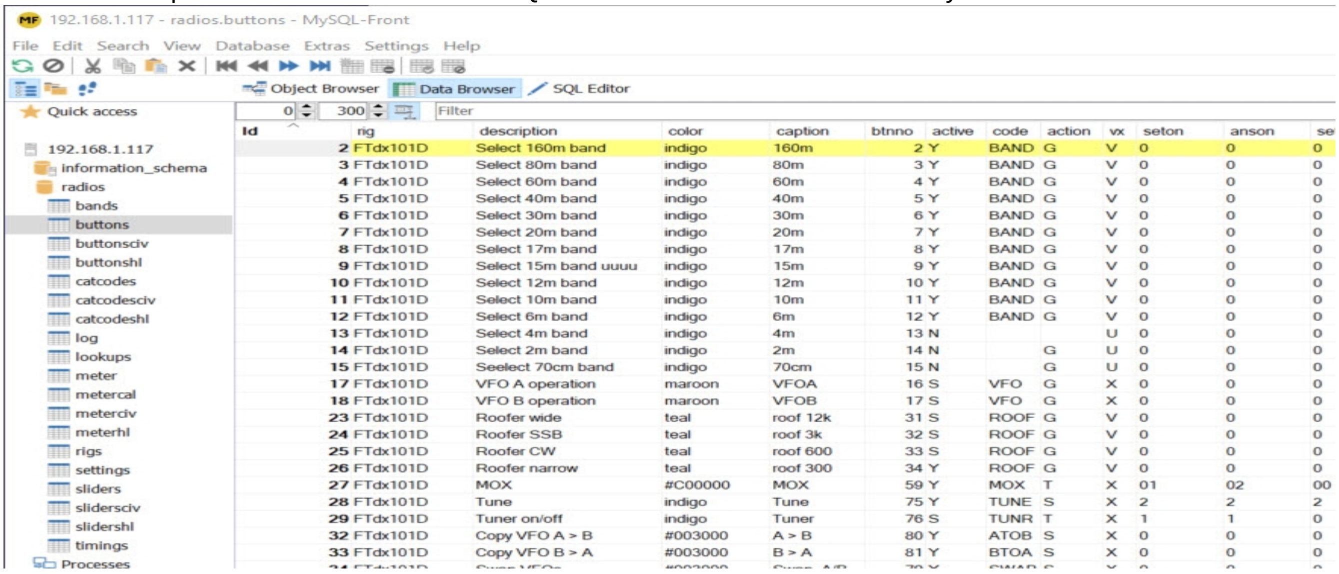
- Using the **table editors of piWebCAT's configuration system.**
  This is extensively documented in this manual.
  Editing is restricted to a selected radio. (eg: The **buttons** table contains control button data for all configured radios but you only see the records for the selected radio. .. which is good!
- Using an **external database editor** such as **MySQL Front** or **HeidiSQL** (free downloads).
  This is discussed below. Editing is of a whole table rather than radio-restricted.

MySQL Front / HeidiSQL ... PC based

You create a login with:
- Host = IP address of RPi, eg: 192.168.1.117
- port = 3306 (This is a standard port for MySQL access)
- user = piwebcat
- password = feline
- database = radios

Below is shown part of my **buttons** table showing data for the FTdx101D.
MySQL Front and **HeidiSQL** allow for **editing** of the database tables and for **backup** and **restore** of tables or the whole database.
The ability to backup the whole database is very important if you are making significant changes.

Using MySQL Front:
Backup: right mouse the **radios** database label top left and then select **Export** .... **to SQL file**.
This will backup the whole database as an SQL file which can be later restored by similar actions.



Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	se
2	FTdx101D	Select 160m band	indigo	160m	2	Y	BAND	G	V	0	0	0
3	FTdx101D	Select 80m band	indigo	80m	3	Y	BAND	G	V	0	0	0
4	FTdx101D	Select 60m band	indigo	60m	4	Y	BAND	G	V	0	0	0
5	FTdx101D	Select 40m band	indigo	40m	5	Y	BAND	G	V	0	0	0
6	FTdx101D	Select 30m band	indigo	30m	6	Y	BAND	G	V	0	0	0
7	FTdx101D	Select 20m band	indigo	20m	7	Y	BAND	G	V	0	0	0
8	FTdx101D	Select 17m band	indigo	17m	8	Y	BAND	G	V	0	0	0
9	FTdx101D	Select 15m band uuuu	indigo	15m	9	Y	BAND	G	V	0	0	0
10	FTdx101D	Select 12m band	indigo	12m	10	Y	BAND	G	V	0	0	0
11	FTdx101D	Select 10m band	indigo	10m	11	Y	BAND	G	V	0	0	0
12	FTdx101D	Select 6m band	indigo	6m	12	Y	BAND	G	V	0	0	0
13	FTdx101D	Select 4m band	indigo	4m	13	N			U	0	0	0
14	FTdx101D	Select 2m band	indigo	2m	14	N		G	U	0	0	0
15	FTdx101D	Seelect 70cm band	indigo	70cm	15	N		G	U	0	0	0
17	FTdx101D	VFO A operation	maroon	VFOA	16	S	VFO	G	X	0	0	0
18	FTdx101D	VFO B operation	maroon	VFOB	17	S	VFO	G	X	0	0	0
23	FTdx101D	Roofer wide	teal	roof 12k	31	S	ROOF	G	V	0	0	0
24	FTdx101D	Roofer SSB	teal	roof 3k	32	S	ROOF	G	V	0	0	0
25	FTdx101D	Roofer CW	teal	roof 600	33	S	ROOF	G	V	0	0	0
26	FTdx101D	Roofer narrow	teal	roof 300	34	Y	ROOF	G	V	0	0	0
27	FTdx101D	MOX	#C00000	MOX	59	Y	MOX	T	X	01	02	00
28	FTdx101D	Tune	indigo	Tune	75	Y	TUNE	S	X	2	2	2
29	FTdx101D	Tuner on/off	indigo	Tuner	76	S	TUNR	T	X	1	1	0
32	FTdx101D	Copy VFO A > B	#003000	A > B	80	Y	ATOB	S	X	0	0	0
33	FTdx101D	Copy VFO B > A	#003000	B > A	81	Y	BTOA	S	X	0	0	0

2.2 Window size and positioning

The image below is piWebCAT's main window configured for my FTdx101D, together
with the three popup windows: log, memory selector and extra buttons.

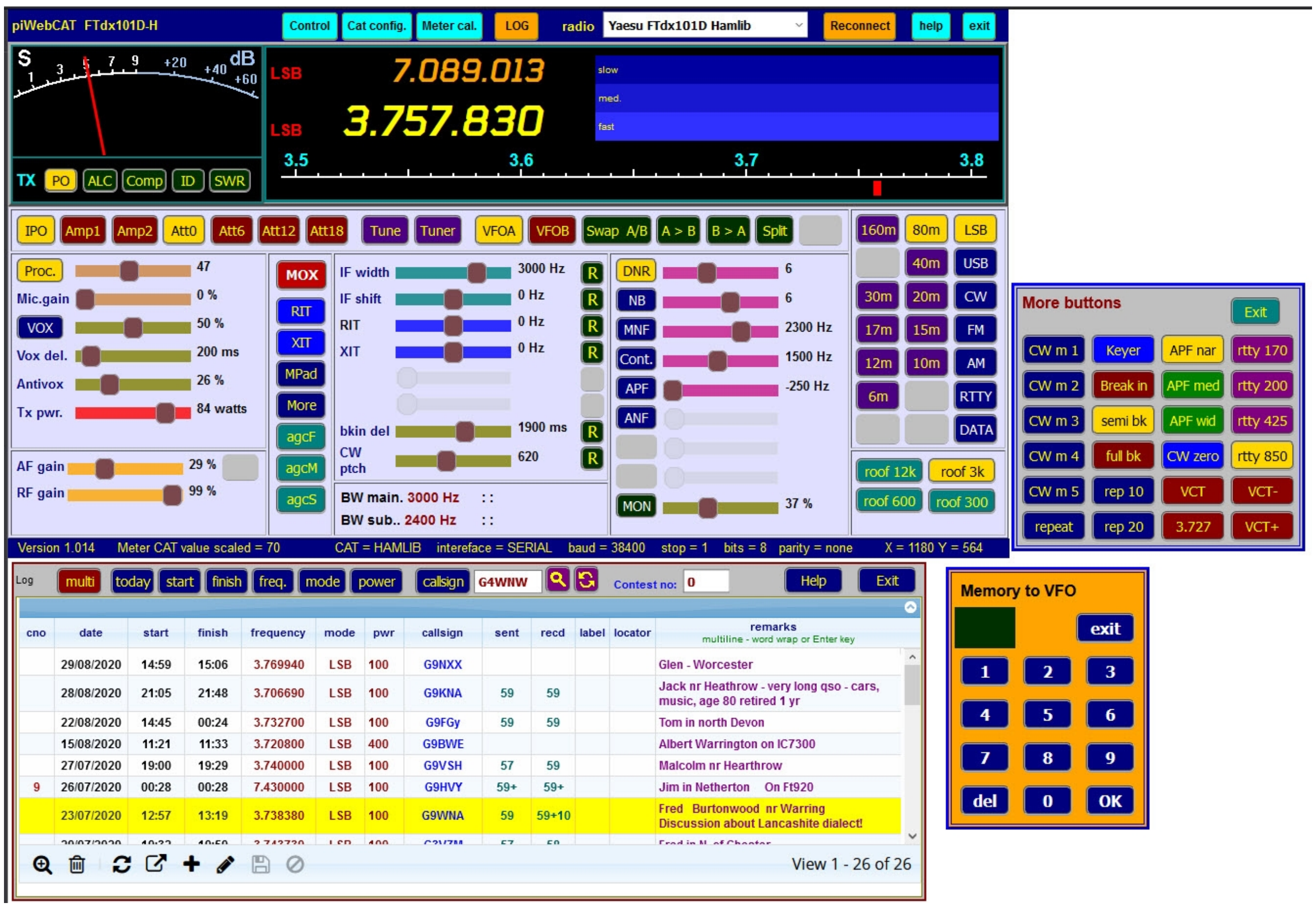
The application starts with the main window only.
This is of fixed size: 1020 x 545 pixels.
It therefore fits within the display limits of modern 1200 x 800 tablets and older 1024 x 768 monitors.

Positioning - memory and button popup windows.
The small key pads (memory select and extra buttons) popup within the main window and can be be used at their popup positions. They are mouse-draggable and so can be moved outside the main window as shown. (Not usually done with the memory keypad because it disappears as soon as you make a selection)

Positioning: log popup window.
Clicking the top bar **log** button displays the log below the main window.
The log window is not resizeable but its size can be pre-determined as logX and logY in the **settings** table.
It can thereby be tailored to your display. There is therefore, for example, plenty of room for a usable log to be located below or alongside the main window on a standard 1920 x 1080 pixel HD monitor.

The three popup windows are in fact all part of the same main control web page.
They are created when the main window is created and simply initially hidden until activated.
This means for example, that the 24 extra buttons can be identical in operation and configuration to the the 66 main window buttons. Furthermore, the station log window has full access to all the radio parameters to facilitate auto insertion of frequency, mode and RF power into the log.

Note that the three popup windows must remain within the web-browser window.
Therefore, to achieve the display below, I first must drag the web browser window to a suitable size to accommodate all four piWebCAT windows.



Version 1.014 Meter CAT value scaled = 70 CAT = HAMLIB intereface = SERIAL baud = 38400 stop = 1 bits = 8 parity = none X = 1180 Y = 564

cno	date	start	finish	frequency	mode	pwr	callsign	sent	recd	label	locator	remarks multiline - word wrap or Enter key
	29/08/2020	14:59	15:06	3.769940	LSB	100	G9NXX					Glen - Worcester
	28/08/2020	21:05	21:48	3.706690	LSB	100	G9KNA	59	59			Jack nr Heathrow - very long qso - cars, music, age 80 retired 1 yr
	22/08/2020	14:45	00:24	3.732700	LSB	100	G9FGy	59	59			Tom in north Devon
	15/08/2020	11:21	11:33	3.720800	LSB	400	G9BWE					Albert Warrington on IC7300
	27/07/2020	19:00	19:29	3.740000	LSB	100	G9VSH	57	59			Malcolm nr Hearthrow
9	26/07/2020	00:28	00:28	7.430000	LSB	100	G9HVY	59+	59+			Jim in Netherton On Ft920
	23/07/2020	12:57	13:19	3.738380	LSB	100	G9WNA	59	59+10			Fred Burtonwood nr Warring Discussion about Lancashite dialect!

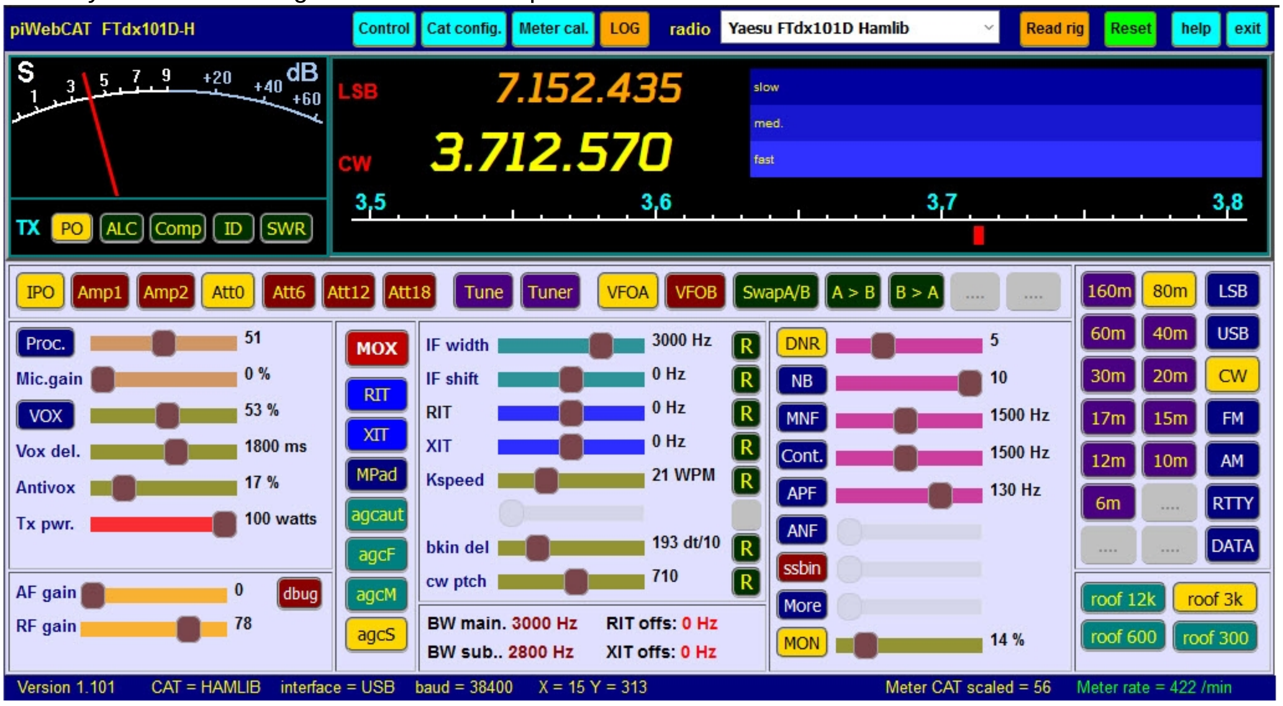
View 1 - 26 of 26

Dragging on a tablet with finger (or Bluetooth or OTG USB mouse)

On an Android or Apple tablet computer (or phone), windows are finger-drag positionable.
This includes piWebCAT's main window. *But the main window has a finger-drag tuner.*
After some experimentation and internet research, I managed to resolve this potential conflict.
Now, if you drag your finger within the tuning window, or over a slider control, the window itself doesn't move. If you drag the window elsewhere it does move.
... So you can separately both position the window and use the controls.

2.3 piWebCAT - operational notes - buttons and sliders

I use my FTdx101D configuration as an example.


piWebCAT FTdx101D-H
Control
Cat config.
Meter cal.
LOG
radio
Yaesu FTdx101D Hamlib
Read rig
Reset
help
exit
S 1 3 5 7 9 +20 +40 +60 dB
TX PO ALC Comp ID SWR
LSB
CW
slow
med.
fast
3,5 3,6 3,7 3,8
IPO Amp1 Amp2 Att0 Att6 Att12 Att18 Tune Tuner VFOA VFOB SwapA/B A > B B > A ....
MOX RIT XIT MPad agcaut agcF agcM agcS
dbug
DNR NB MNF Cont. APF ANF ssbin More MON
160m 80m LSB 60m 40m USB 30m 20m CW 17m 15m FM 12m 10m AM 6m .... RTTY .... DATA
roof 12k roof 3k roof 600 roof 300
R

7.152.435
3.712.570

Proc. 51
Mic.gain 0 %
VOX 53 %
Vox del. 1800 ms
Antivox 17 %
Tx pwr. 100 watts
AF gain 0
RF gain 78

IF width 3000 Hz
IF shift 0 Hz
RIT 0 Hz
XIT 0 Hz
Kspeed 21 WPM
bkin del 193 dt/10
cw ptch 710

BW main. 3000 Hz RIT offs: 0 Hz
BW sub.. 2800 Hz XIT offs: 0 Hz

DNR 5
NB 10
MNF 1500 Hz
Cont. 1500 Hz
APF 130 Hz
MON 14 %

```
Version 1.101    CAT = HAMLIB   interface = USB   baud = 38400   X = 15 Y = 313             Meter CAT scaled = 56   Meter rate = 422 /min
```

Buttons and Sliders

Configuration is discussed in detail elsewhere ... but note the results of configuration:

- **Many buttons are in groups**: eg: band, mode, IPO, Attenuators, roofers, VFO A/B.
- **Some buttons toggle on/off**, eg: DNR, NB, MNF, APF, MON, Vox.
- **Some buttons are single action**, eg: swap A/B, A>B, B>A, +25kHz, -25kHz
- Two **buttons launch popups**,ie; **MPad** (memory keypad) and **More** (24 more buttons)
  (Note that even these have user configured button choice, caption and colour).
- A button can be confiigured to switch to a specified memory channel (eg: **M17** in the examples)
- **Sliders** are configured to match the range of values on the radio.
  Example: **IF shift** ... this is user configured for centre zero with range -1200Hz to +1200Hz.
  The **R** buttons reset a default value .... which for IF shift is zero (centre)
- The five Tx meter buttons select the meter value to be displayed on transmit.
- **MOX** toggles transmit/receive ... and also lights up on PTT or radio MOX.

The detailed configuration needs to reflect the operational structure of the radio.
eg: In the FTfx101D,

- Most receiver button and slider parameters have different values between the two receivers (A and B). (*This is not so all radios*.)
- Some settings change on band change.
- We only have one set of controls ***which must reflect the current VFO's settings***.

Loading all the settings can take 4-5 seconds. During this transition time, the controls are inactive.

The commonest need for a rapid transition is on VFO A/B switching (with or without band change).
piWebCAT provides for this by storing the most recent slider and button states for each VFO.
eg: VFO A selected - switch to 40m band - 5 second delay loading settings.
then select VFO B which is on 20m band - 5 second delay (unless previously selected)
Thereafter, VFO switching is rapid using the stored 40m data for VFO A and the 20m data for VFO B.

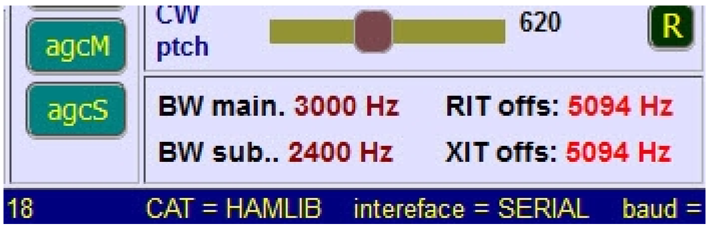
Control, Read rig and Reset buttons

- **Control** Switches to or completely redraws and restarts the main page
- **Reset** Restarts the main page program code. Popup windows positions are preserved. (eg: log)
  Control settings are resynchronised to the corresponding rig parameters. (approx 5 sec)
- **Read rig** Control settings are resynchronised to the corresponding rig parameters. (approx 5 sec)

2.4 Text display box - this section is repeated in Hamlib configuration

A late modification was to replace the lowermost slider in the middle section by a text display box.
The box has four text data items, each with a caption in black and data text in a user defined colour.
The four text items appear in the **sliders** / **slidersciv** /**slidershl** tables with slidernos: **51** to **54**.

Each item is essentially a slider caption and a slider text data display without the slider in between them.
The caption is defined by **slider.caption** The **slider.color field** is used for the data text colour.
The **mult**, **divide**, **offset**, **units**, **lookup** and **decpoint** fields act on the data display exactly as for sliders.
min and **max** are unused. (They are used by sliders to match slider travel to min and max data range limits.)

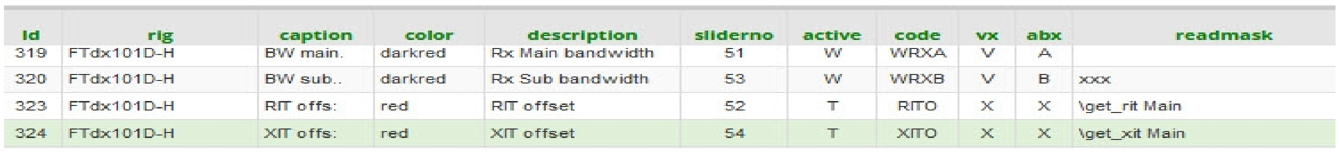


In this example, the **RIT** and **XIT** items, 52 and 54 are synchronised to the data values in the rig.

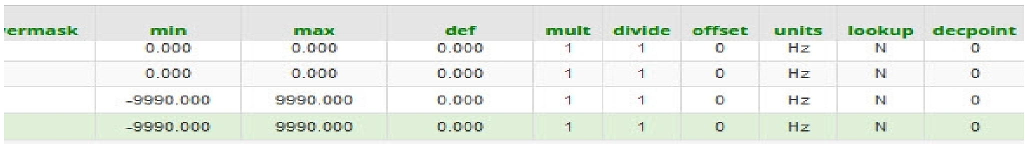
The BW items, 51 and 53 here are using a special Hamlib feature. This is related to the fact that the necessary repetitive Hamlib **mode** readings also automatically return bandwidth.

This example is from my FTdx101-H Hamlib configuration (ie: connecting using the Hamlib **rigctld** daemon.)

The CAT data configuration is the same as for a slider (except **min** and **max** unused)
So please refer to the appropriate slider section for your configuration (ie: ASCII, YAESU5, CIV or HAMLIB)
(ASCII will user **answermask** to match the data. CIV uses binary data has a dedicated communication system.)



id	rig	caption	color	description	sliderno	active	code	vx	abx	readmask
319	FTdx101D-H	BW main.	darkred	Rx Main bandwidth	51	W	WRXA	V	A	
320	FTdx101D-H	BW sub..	darkred	Rx Sub bandwidth	53	W	WRXB	V	B	xxx
323	FTdx101D-H	RIT offs:	red	RIT offset	52	T	RITO	X	X	\get_rit Main
324	FTdx101D-H	XIT offs:	red	XIT offset	54	T	XITO	X	X	\get_xit Main



ermask	min	max	def	mult	divide	offset	units	lookup	decpoint
	0.000	0.000	0.000	1	1	0	Hz	N	0
	0.000	0.000	0.000	1	1	0	Hz	N	0
	-9990.000	9990.000	0.000	1	1	0	Hz	N	0
	-9990.000	9990.000	0.000	1	1	0	Hz	N	0

The above table is split into left and right parts. The unused **setmask** and **answermask** fields are not shown.
(Note that **answermask** would be used with **\send_cmd_rx** for a CAT command that is not supported by Hamlib)

The **BW main** and **BW sub** above are a special Hamlib feature, see below.
*The **RIT** and **XIT** items represent the usual configuration of these text data items for all other level data.*

RIT and XIT offset

The CAT data field here is **readmask** which with Hamlib **rigctld** is simply: **\get_rit Main.**
(See: rigctl documentation)
The offset frequency data is returned as a numeric with range -9990 to +9990.
This needs no scaling and so **mult = 1** and **divide = 1**.
(For AF gain with range 0.000 to 1.000 we might set mult = 100 to scale the displayed value to 0 - 100)

Above, I have used **code=RITO** and **code=XITO**. I could have used any **code** of my choice.
code and **vx** are sent to the client. **code** and **abx** are used on the server. They form the client - server link.
(See: README - vx and abx)

active = T

For a text data item, we set **active = T** as in the above table.
This has the same effect as setting **active = S** (sync) for a slider.
This is essential so that the text item is repetitively updated together with other controls with active = S or T.

Hamlib rigctld - Bandwidth display

In the above text box and associated **slidershl** records, the **BW Main** and **BW Sub** items use a feature that is only available using Hamlib.
(Main and Sub are FTdx101D terminology. Most rigs refer to VFOA and VFOB)

However, you can still display BW Main and BW Sub with other configurations (eg: ASCII) using the configuration technique that RIT and XIT use in the above example.

Hamlib rigctld reads and sets Mode and Bandwidth together in a single command.

rigctld \get_mode Main and \get_mode Sub are configured like any other rigctld command but return current bandwidth as a bonus. So we have regular bandwidth updates without having to set up a specific bandwidth read.

Reading Main (or VFOA) **Mode** and Sub (or VFOB) **Mode** from the rig each have a dedicated repetitive process.
The repetition intervals for these reads are set in the timing table using fields:
modecur (currently selected VFO) and **modezz** (sleeping VFO)

Mode data read events are set up in the **buttonshl** and **catcodeshl** tables.
When the bandwidth data is returned with mode data, piWebCAT checks for **sliderhl** text item records with **active=W** and with **code** field set as follows:

- With Hamlib **--vfo mode** use **code=WRXA** for Main/VFOA
  and **code=WRXB** for Sub/VFOB.
- When **not using --vfo mode** (eg: Icom) use **code = WRXX**.

Hamlib --vfo mode is determined by **vfomode=Y** or **vfomode=N** in the **rigs** table entry for the rig.

The bandwidth data display is then updated.

With Hamlib, operating mode and receiver bandwidth share the command \get_mode.

The returned vfo identifier of **\get_mode** is **A** or **B** for vfomode = Y and X for vfomode = N.
These identifiers originate from the **abx** field of the mode button field in the **catcodeshl** table.
They must be appropriately set:
- For **vfomode = Y**, there will be two entries, one with **abx =A** and one with **abx = B**.
- For **vfomode = N**, there will be a single entry with **abx = X**.

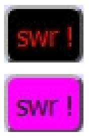
2.5 Indicator controls - periodically updated

piWebCAT has a total of **27 sliders** and **90 buttons**.
A startup, the state or buttons and the positions and text values of slider are updated from the radio.
This also occurs on band change and on use of the Reconnect button.
Thereafter, there is no regular update of the controls. Regular update is not necessary if piWebCAT is controlling the radio (because the controls are changed on piWebCAT and the radio responds).

There are, however, options to have selected controls updated on a regular basis.
For each control, the **active** field has options: **Y**, **N**, **S** and **L** as follows:

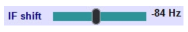
Buttons

- **N** Inactive - grayed out.
- **Y** Normal active state - no regular state updates.
- **S** 'Sync' - periodically updated to the associated radio parameter.
  Sync buttons can be mouse-clicked as active buttons.
- ***L*** *'LED' - as Sync but with a different appearance:*
  OFF *state is black + red text. ON state is a bright colour of your choice + black text.*
  (ie: Your configured colour specifies the ON colour rather than the usual OFF colour)
  LED buttons buttons are not active controls, ie: they do not respond to mouse clicks.



Sliders

- **N** Inactive, grayed out.
- **Y** Normal active state - no regular state updates.
- **S** 'Sync' - the slider position and text data are periodically updated to match the radio parameter.
- **L** As 'Sync' but with a different appearance
  (Narrow black slider thumb on teal background)



- **T** One of the four text box data items
  (update as Sync items)
- **W** A special Hamlib-only IF width display.

The controls remain active in controlling the radio if configured to do so.

Sync timing configuration

The **update period** is set by the **sync** field in the radio's **timing** table record.

Example: The **sync** field is set to 300ms and you have five buttons and three sliders set to **S** or **L**.
The update process steps around the eight controls every 300ms and so each item updates every 2400ms.
This facility was not thought of during initial development with the FTdx101D and the IC7000.
Nearly all CAT controls on these radios are read / write and so read-only indicators were not needed.
The facility was first introduced when investigating the CAT needs of the Yaesu FT920.

My FT920 configuration has slider controls (all set to L) for:
NR level, IF Lo cut, IF hi cut, IF shift, Speech proc. level and Squelch.
Rx clarifier control has a read / write configuration and is also updated by having active = S.

The FT920 has some issues:

- Its RS232 baud rate is rather slow at 4800 baud.
- Some of the parameters share a common CAT command that reads a large block of data.
- The clarifier offset is two bytes in the middle of a 28 byte returned data block for VFOs A and B.

For the FT920 and other YAESU5 radios, there is therefore a 'rigfix' option whereby a single data block read can serve multiple controls.

2.6 Tuning controls and RIT / XIT

Band switching: The band buttons change band on the radio which switches to a designated frequency for the new band. The new band's choice of initial frequency is a radio feature (Usually from last usage.) piWebCAT is continually monitoring the radio's frequency.... of both VFOs if possible.
(The IC7000 only appears to have a CAT command to read the ***current*** VFO. piWebCAT therefore has a **rig** table option to suppress the display of the background VFO's frequency)

piWebCAT with change band display (band tuning scale) on detecting a changed current VFO frequency that is ***within the limits of a new band.***
piWebCAT does not change band just because the radio's frequency has left the old band.
If you tune the radio beyond the band edge, the piWebCAT band display will only change band when you reach a new band.

Note that band edges are in the **bands** database table and so can be changed.
However, the band limits of the tuning scale are, by necessity, fixed in the code.
Thus, if a band were to be extended, you could extend the band limits so as to have the correct band display selection behaviour. Tuning into the extended area would simply go off-scale.

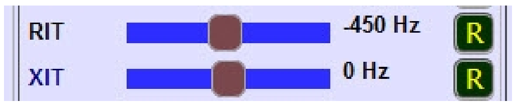
The older Yaesu radios such as FT920, FT100MP MkV (YAESU5 option) do not have a band-select command.
Each band select button is therefore configured with your choice of start frequency for each band.
On band change, the button sends the frequency to the radio, piWebCAT then responds to the radio's frequency change by switching its tuning scale display to the chosen band.

Coarse frequency setting - click the tuning scale - the frequency changes and the marker moves.

Fine tuning

Use the blue tuning band with horizontal mouse (or finger) drag OR with the mouse thumbwheel.
The tuning band has fast, medium and slow lanes.
The response of each lane is user configurable in the settings database table.
(ie; Hz / pixel for mouse/finger drag and Hz/click for the thumbwheel)

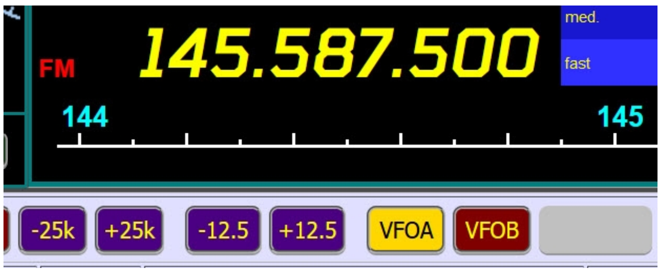
RIT and XIT slider - use of mousewheel



RIT and XIT control can be assigned to sliders as shown above.

Using **Firefox** as a the web browser, the **mousewheel** can be used to tune in fine steps of 1/400 th of the span.
In the case of the FTdx101D shown above, that is 1/400 th of -9990 to to +9990 Hz = 50Hz per step.
With the mouse pointer over the slider, the mousewheel is clicked and thereafter can be used for fine tuning.
Browsers: Chrome and Opera do not provide this feature at the time of writing. This may change !

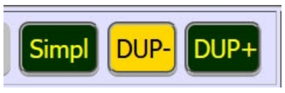
Frequency up/down buttons



The user can program any button as a frequency **Up** or **Down** button.
Here, I have four such buttons in my IC7000 Hamlib configuration. They are: **+/- 25 kHz** and **+/- 12.5 kHz**.

The first click moves the frequency to the next multiple of 12.5 kHz or 25 kHz. Thereafter the buttons step at the specified interval.

Duplex / simplex switching



For repeater operation, I configured the three buttons shown for the IC7000.
The rig doesn't have the means to assign repeater operation to specific channels and so Duplex - or + has to selected. piWebCAT's buttons here are much faster and easier to access than the corresponding control sequence on the rig.

2.7 piWebCAT station log

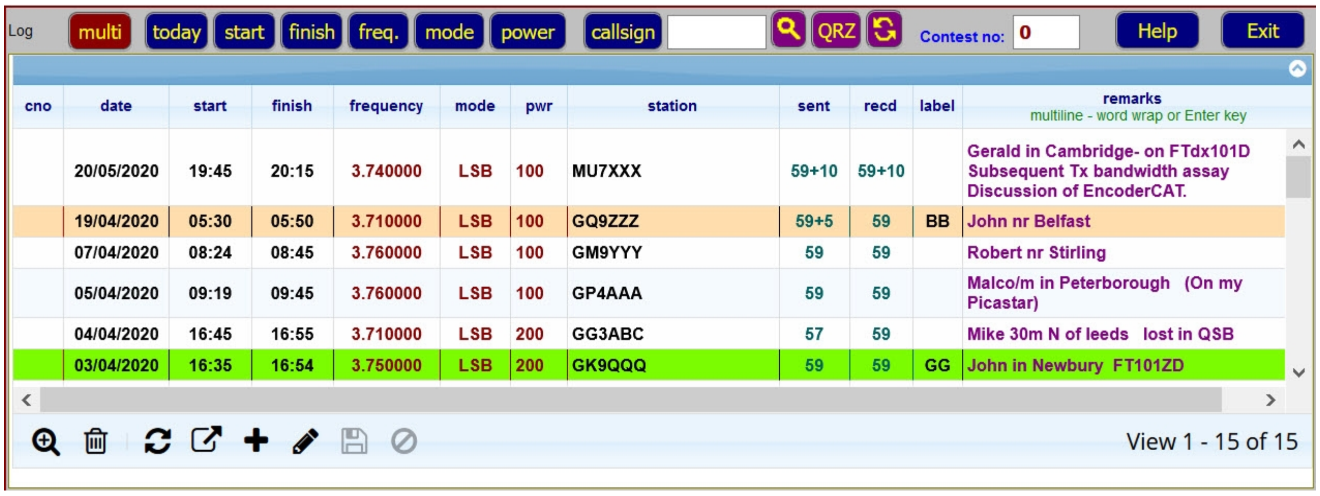
Note that correct automatic time entry into your log needs an internet connection or a RPI real time clock module (piHut £5).

The RPi stores current time on power down.
Without internet or real time clock, it's software clock will restart with the stored power down time.

Note that the integrated logging system does not work without a working connection to a rig.
An alternative stand-alone version is therefore provided (with access to the same data.) See below.

The log window is displayed by the **Log** button on the top button bar.
It appears below the main window but can be dragged to other positions *within the browser window*.
The log is stored in table **log** in the RPi MySQL database.
It has the following features:

- One **single button click** enters: date, start time (UTC), frequency, mode, power, optional contest number and callsign if first entered in the top line box as below.
- **Date picker** and **time sliders** for 'manual' data entry.
- Optional **multiple lines** with word wrap or enter key in the remarks column (automatically expands the row).
- Searchable **label column** with three specific labels options to highlight the row in a background colour.
- **Search button** on button toolbar - search option on all columns.
- **Callsign search** button on top button bar. Search on the callsign displayed in the box.
- **QRZ.com** button will launch the qRZ.com website for the displayed callsign. (QRZ login retained after login) Clicking on any QSO record will place the callsign in this top box.
- **Log window width and height** predefined by logX and logY fields in database **settings** table.
- **Export to CSV** button launches in Excel for printing etc (Minor excel formatting changes needed for word wrap)
- **Backup and restore** of log using third party MySQL editor (eg: MySQL Front ...free download)



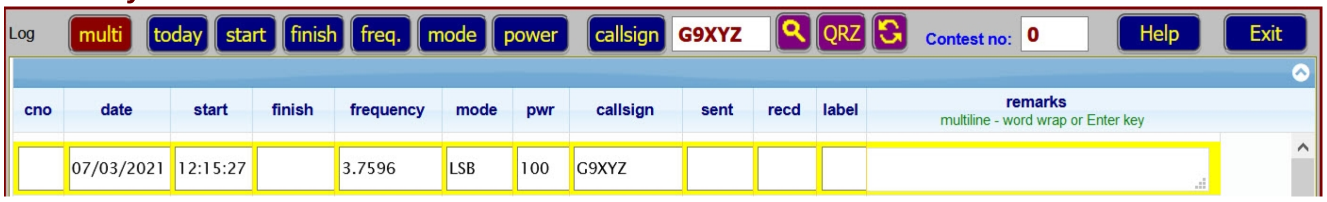
cno	date	start	finish	frequency	mode	pwr	station	sent	recd	label	remarks multiline - word wrap or Enter key
	20/05/2020	19:45	20:15	3.740000	LSB	100	MU7XXX	59+10	59+10		Gerald in Cambridge- on FTdx101D Subsequent Tx bandwidth assay Discussion of EncoderCAT.
	19/04/2020	05:30	05:50	3.710000	LSB	100	GQ9ZZZ	59+5	59	BB	John nr Belfast
	07/04/2020	08:24	08:45	3.760000	LSB	100	GM9YYY	59	59		Robert nr Stirling
	05/04/2020	09:19	09:45	3.760000	LSB	100	GP4AAA	59	59		Malco/m in Peterborough (On my Picastar)
	04/04/2020	16:45	16:55	3.710000	LSB	200	GG3ABC	57	59		Mike 30m N of leeds lost in QSB
	03/04/2020	16:35	16:54	3.750000	LSB	200	GK9QQQ	59	59	GG	John in Newbury FT101ZD

View 1 - 15 of 15

Above is a section of my log (callsigns scrambled!) (Not a lot of activity in April/May 2020 - too busy creating piWebCAT !!)
Note:

- The **highlighted entries**. Highlight is controlled by the **label** column.
  Labels can be whatever you want and are searchable.
  Three are reserved: BB = brown highlight (illustrated above), GG = green highlight, PP = purple highlight.
- The **multiline remarks column**. This supports both word wrap and 'Enter' key for a new line.
- **Contest number** edit box: If zero, no contest number is entered. If non-zero, number is auto-entered to the **cno** column and then incremented for next QSO.
- The **locator** column. This is *optionally displayed.*
  The **locW** field of the **settings** table specifies the width of the **locator** column. The range is 0 - 140 pixels.
  If locW is less than 46 then the heading is abbreviated to 'loc'.
  If locW is zero, then the column is hidden in the log display
  (The locator field persists in the database and no data is lost)

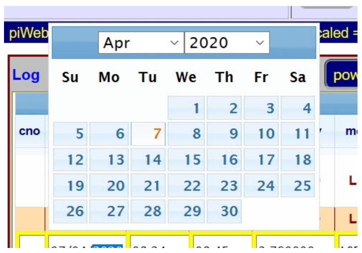
Data entry



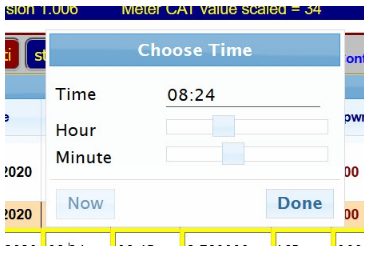
cno	date	start	finish	frequency	mode	pwr	callsign	sent	recd	label	remarks multiline - word wrap or Enter key
	07/03/2021	12:15:27		3.7596	LSB	100	G9XYZ				

Most of the time, I use the **multi** button.
The best procedure is to type the callsign into the button-bar callsign text box.
This automatically displays upper case letters, whereas typing into the grid needs the shift or caps-lock key.
Click the bottom bar **+** button to display the data entry row as above and then click the top bar **multi** button.
This will automatically enter the data shown above (+ optional contest number).
Then click the button.
At the end of the QSO, click the row to edit and click the **finish** button to enter time now (UTC) as finish time.
Click .

Note that the **start**, **finish**, **freq.**, **mode, power** and **callsign** buttons can enter these data items separately.



Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		



Time 08:24
Hour
Minute

For 'manual' data or time entry, a date picker and time sliders are provided.
These are not normally needed unless you are entering retrospective data

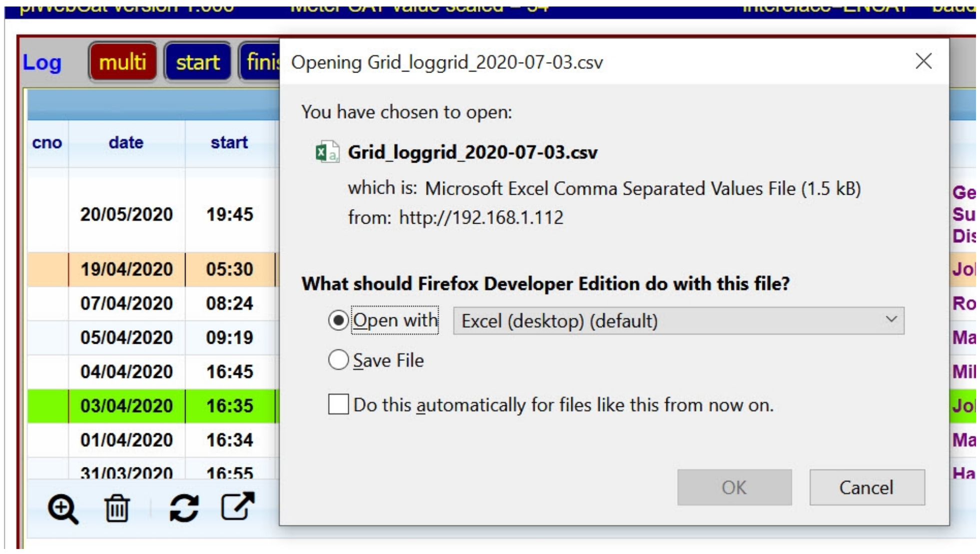
Use the button to cancel data entry.

Editing the log

Single clicking a row displays it in the edit / new entry format as above
(*and copies the callsign to the top callsign box.*)
The editing facilities (including the auto-entry buttons) are as for a new entry.
Use the button to cancel edits.

Import, export. backup.

The button exports as a CSV file (comma delimited) and offers to launch it in Excel.



Opening Grid_loggrid_2020-07-03.csv

You have chosen to open:

Grid_loggrid_2020-07-03.csv

which is: Microsoft Excel Comma Separated Values File (1.5 kB)
from: http://192.168.1.112

What should Firefox Developer Edition do with this file?

(●) Open with Excel (desktop) (default)
( ) Save File
☐ Do this automatically for files like this from now on.

cno	date	start
	20/05/2020	19:45
	19/04/2020	05:30
	07/04/2020	08:24
	05/04/2020	09:19
	04/04/2020	16:45
	03/04/2020	16:35
	01/04/2020	16:34
	31/03/2020	16:55

MySQL Front can be used to export and import the whole log in **CSV format** or as an **SQL database script**.
Export to SQL is an excellent way to create a backup.

Log sort order

The log is sorted on fields **date** and then **start** (time)

These fields are held in the database in standard MySQL date and time format.
You can view them with MySQL Front:
eg: **2020-07-03   12:34:23**

piWebCAT converts these to **03/07/2020   12:34** for display.

The database combination of **202-07-03 12:34:23** is used for correct time based sorting.

The presence of the seconds data ensures that contest records at a rate of more than one per minute are correctly ordered !!

Searching

The button provides search option on any column.

The button searches for the callsign in the adjacent text box to generate a display containing only the QSOs with that callsign,

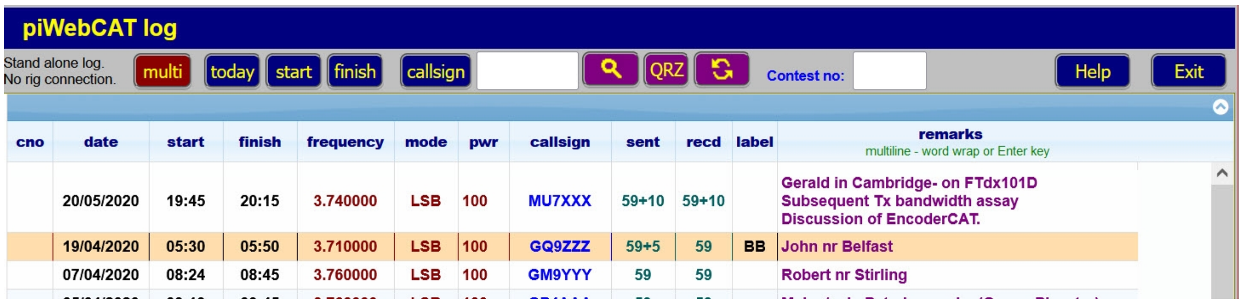
The adjacent button restores the full log display.

Stand-alone logging system

The integrated logging system as described above is part of the main piWebCAT web page.
If there is no working connection to a rig, then the logging system will not function.
An alternative stand-alone version is therefore provided.
This is accessed by:

- The **Log only** button on the start page.
- Using a URL such as **192.168.1.117/log.php**   where 192.168.1.117 is the IP address of the RPi.

In this version, the **Multi** button does not auto-insert frequency, mode nor power.
The corresponding buttons are absent.



cno	date	start	finish	frequency	mode	pwr	callsign	sent	recd	label	remarks multiline - word wrap or Enter key
	20/05/2020	19:45	20:15	3.740000	LSB	100	MU7XXX	59+10	59+10		Gerald in Cambridge- on FTdx101D Subsequent Tx bandwidth assay Discussion of EncoderCAT.
	19/04/2020	05:30	05:50	3.710000	LSB	100	GQ9ZZZ	59+5	59	BB	John nr Belfast
	07/04/2020	08:24	08:45	3.760000	LSB	100	GM9YYY	59	59		Robert nr Stirling

2.8 Learning guide - Introduction, hardware setup

This guide makes use of three generic radios configured in the database on the supplied SD card.
They are selectable using piWebCAT's drop down radio selector.
The three radios and this guide were a late addition to this manual.
The guide repeats information that is documented elsewhere in a progressive learning presentation.
Whatever rig you are using, please make sure you read section 2.10 Icom --vfo mode

three configured radios have the following features:

- They use piWebCAT's **HAMLIB** setting which uses Hamlib's **rigctld** system.
  Hamlib **rigctld** converts generic Hamlib CAT commands into the commands for the selected radio.
- Initial configuration requires only the Hamlib radio code, USB or SERIAL selection and the baudrate.
- The three configurations evolve A to B to C with progressive addition of more controls.
  Thus A is very simple with only tuning, band and mode selection, IF width and MOX.
  C is well featured but with controls limited to those that are common in modern radios.
- My idea is that you gain progressive familiarity with piWebCAT using A to C and then consider adding your own rig-specific CAT commands using the remaining large number of unused controls.

In describing the setting up of these example radios, numerous potential side issues and explanations arise.
Some of these are discussed here. Many are referenced to relevant sections of this document / website in order avoid repetition and to produce a fairly compact guide.

Hardware setup

You need to start with the correct hardware setup for your radio.
See section 13.1 **Serial Pi Zero** and **G3VPX piWebCAT interface card**.
See section 13.2 **piWebCAT card schematic**. Ensure link is set correctly for RS232 or Icom CI-V.

Raspberry Pi micro D card

The supplied SD card has preinstalled: Raspbian operating system, Apache2 web server. PHP language, MariaDb (MySQL) database server, Mumble VOIP server and client, Hamlib rigctl/rigctld, an FTP server (pure-fptd) for code upload, VNC server and the piWebCAT code and database.

RPi keyboard / mouse /monitor RealVNC

Either plug in a USB keyboard and mouse and a monitor OR use VNC to control from your PC.
VNC installation on PC: Download and install **VNC viewer** from ReadlVNC (free).
From VNC Viewer: From menu - **File** - **Add connection.**
Add the RPi IP address as VNC server (192.168.1.117) Give it a name . Click OK. Then click the icon.
Username = pi, password = feline. Check the save-password box.

Note that in normal operation, the RPi can be treated as a 'black box', both for piWebCAT and for the Mumble audio link. Likewise for CAT configuration changes that are made via piWevCAT. However, keyboard access to the RPi user interface may be needed during configuration, eg: for testing Hamlib rigctrl commands in the RPi terminal window (and finding your Hamlib code)

RPi IP address

The RPi has been configured to connect via Wifi (wlan0) and via wired internet (eth0).
The SD card is supplied with IP address 192.168.1.117 assigned to both wlan0 and eth0.
If you need to change the IP address, then with RPi terminal: use `sudo nano /etc/dhcpcd.conf` and simply find the two instances of 192.168.1.117 and change to an IP address of your choice.
Then save with `Ctlr X, Y, Enter`. Then `sudo reboot`.

Note: I have supplied the SD card with **Mumble VOIP** audio up and running connected to 192.168.1.117.
If you change your IP address, you need to disconnect Mumble client on the RPi from Mumble server on the RPI. Reconnect with the new IP address: Accept the dodgy certificate message by clicking YES.

Safety backup of configuration MySQL database

You will eventually be modifying the configuration database on the supplied SD card.
It is a good idea to begin by backing up the whole database (ie: all configured radios and the station log).
Please download and install **MySQL Front**. This is a free of charge database editor and SQL toolbox.
Section 3.13 gives instructions in how to use MySQL Front to backup the **radios** database.

2.9 Learning guide: Hamlib - rigctl rigctld

Hamlib is supplied ready installed on the micro SD card.
Hamlib is 12 years old but is still steadily improving and evolving.
This is particularly so for new radios where there may be bugs in the system.

I developed piWebCAT on the FTdx101D, IC700 and then FT920.
Then I discovered Hamlib. The Hamlib FTdx101D interface was very new and full of bugs.
I needed it to work properly for piWebCAT.
Michael Black, W9MDB was doing much of the development ... and he needed it work properly.
So we worked together for while over the internet to fix all the problems.
I downloaded his latest Hamlib code many times.
Section 14.5 describes how to do this: 14.5 Installing / updating Hamlib on the SD card

You need the Hamlib code for your radio.
My configurations show 3060 for the IC7000 and 1040 for the FTdx101D).
Go to the RPi terminal window and enter `rigctl -l` (lower case L).
This will list all the radios and their codes. Make a note of your radio's Hamlib code.

Hamlib rigctl and rigctld - documentation -commands

Documentation

Download this from the following URL *and print it for easy reference.*

https://manpages.debian.org/unstable/libhamlib-utils/rigctl.1.en.html

You will see references to **rigctl** and rigctld**.**

A detailed explanation of using Hamlib with piWebCAT is given in sections 8
See section 8.1 Hamlib - introduction and 8.2 onwards

rigctl

The connections to the radio via the **rigctl** API (Application Programming Interface)
rigctl translates its generic CAT commands into the corresponding commands for the selected radio.
eg: For a rig DNR (Digital Noise Reduction) level of 7
Yaesu FTdx101D DNR range 1 to 15 CAT = RL00**7**;
Icom IC7000 DNR range 0 to 15 CAT = 0xFE 0xFE 0x70 0xE0 0x1A 0x05 0x01 0x14 0x**07** 0xFD
rigctl: \set_level NR VFOA 0.5 (--vfo mode) or **\set_level NR 0.5** (non --vfo mode)
(range is 0.000 - 1.000) translates to DNR = 7:

The use of --vfo mode is discuused in the following section

rigctl commands can be tested in the RPi terminal window

You activate this at the command prompt by: `rigctl -m 1040 -s 38400 -r /dev/ttyUSB0 --vfo`
where:

- `1040` is the Hamlib radio code (FTdx101D in this case)
- `38400` is the baud rate as set on the radio
- `/dev/ttyUSB0` is the connection for USB (or `/dev/ttyAMA0` for serial connection)
- `--vfo` enables VFO mode which I have used in the three piWebCAT configurations.

You can then enter **rigctl** commands as documented in the manual.
All the commands have a long and a short form. The long form must be prefixed by **** .

Try the activation command with your radio and baudrate.
Then enter: `\set_freq VFOA 3730000` to set a frequency (short form is `F VFOA 3730000`)

rigctld

rigctld is the rigctld interface to which piWebCAT connects.
It provides a socket to which server PHP code can connect and send the rigctl commands.
You don't have to deal with it. You just set up the correct codes in the configuration database.

VFOA / VFOB or Main / Sub   Transceivers-H-A, B and C - modern Yaesu rigs.

A progression of three developing learning configurations is supplied for dual vfo / dual receiver transceivers which are supported by Hamlib   **--vfo** mode.

They are Transceiver-H-A, Transceiver-H-B and Transceiver-H-C.

These three learning configurations in **--vfo** mode are supplied with VFOs specified as VFOA and VFOB.
They were developed using my FTdx1010D.

The FTdx101D (and some other modern Yaesu rigs) use Main / SUB rather than VFOA / VFOB.
In the FTdx101D, only the **\set_vfo** and **\get_vfo** commands were found to be critical in this respect.

Because of this, the VFOA and VFOB parameters for the these commands in the **buttonshl** table were temporarily replaced with Main and Sub.

They have been changed back to VFOA / VFOB before issuing the SD card

If you want to use these three Hamlib configurations with a modern Yaesu rig, then please use MySQL Front to run the supplied SQL script: **VFOA_VFOB_to_Main_Sub_Hamlib.sql.**

The script operates on tables: **buttonsh**l, **catcodeshl**, **slidershl** and **meterhl**.

You need to edit the script (eg: with Notepad) to change the rig name.

You would need to run the script on each of Transceiver-H-A, Transceiver-H-B and Transceiver-H-C
 (editing the name for each one of these).

See section 3.14 MySQL commands and scripts   Section 3.15 Useful MySQL scripts

2.10 Learning guide - Icom transceiver issues

piWebCat was developed for Icom transceivers on a IC7000, which is 17 years old.
I have not tested piWebCAT on a modern Icom transceiver.
I have therefore not used a USB connection with Icom.

Examination of modern Icom CAT manuals shows that the CAT system is essentially unchanged.
The number of functions has greatly increased and there is extensive renumbering of subcommands.
This learning guide uses Hamlib's generic command set which translates into the correct CAT commands and thereby avoids the detail and complexity of the Icom CI-V system.

Furthermore, the CI-V interface circuitry appears essentially unchanged.

Hamlib rigctl --vfo mode.

Examples:
To read noise reduction level in non --vfo mode, the command is **`\get_level NR`**

To read noise reduction level in --vfo mode, the command is **`\get_level VFOA NR`** (or ..VFOB)

So we use VFO mode for radios whose CAT commands can access or need to access parameters of a specific 'VFO' or receiver.
eg The FTdx101D has two completely separate receivers and CAT commands can address them individually, irrespective of which receiver is currently in use.
In --vfo mode, all commands must have a VFOA or VFOB etc parameter. *This is so, even if it is just a dummy parameter because the command doesn't really need it (eg: RF power setting)*

I quickly realised that the Icom IC7000 command set only addresses the currently active VFO.

I was later fairly astonished to find that modern Icom radios do not appear to have CAT access to the background VFO.

With the Yaesu rig, I display both VFO frequencies in piWebCAT. This is done by periodically reading the frequencies (and modes) from the rig. The read intervals are user defined in the **timings** table and can be made less frequent for the background VFO.
Hamlib commands **`\get_freq VFOA`** and **`\get_freq VFOB`** read the frequencies.
(Hamlib translates them into simple Yaesu commands: **`FA;`** and **`FB;`**)

I can't easily do this with Icom because I can only read the active VFO frequency. I cannot read the background VFO frequency. The same applies to VFO associated parameters such as NR, NB etc.

I discussed this with Mike, W9MDB who is doing much of the Hamlib development. He confirms my observations and says that this situation applies to several other rigs as well as Icom.
I quote him:
"...There are a lot of rigs that can't read the "other" vfo without swapping.
That's why you don't see any programs polling both vfos".
.... *Please correct me if I am missing something here!!*

*piWebCAT **does** support dual VFOs if the rig's CAT system provides the necessary access.*
Likewise - Hamlib's --vfo mode is available for those rigs that can use it.

With regard to other receiver parameters (NR, NB, width etc): I do not know if modern Icom radios hold a full separate parameter set for each receiver ... but in their CI-V manuals, I can only see a CAT access route to the the active receiver. If it isn't there, then Hamlib cannot implement it.

With the IC7000, if use --vfo mode and send a command to read frequency and other parameters from the background VFO, then Hamlib momentarily switches the background VFO to achieve this.
This produces intolerable oscillatory band changing between VFOs.
There is no point having the background VFO displayed in CAT software if it not repetitively updated.

So with Icom I suggest NOT using VFO mode.

With regard to the list of 250 radios supported by Hamlib, I cannot tell you which give CAT access to the background VFO. You need to experiment with Hamlib rigctl at the RPi command line.

This learning guide and --vfo mode

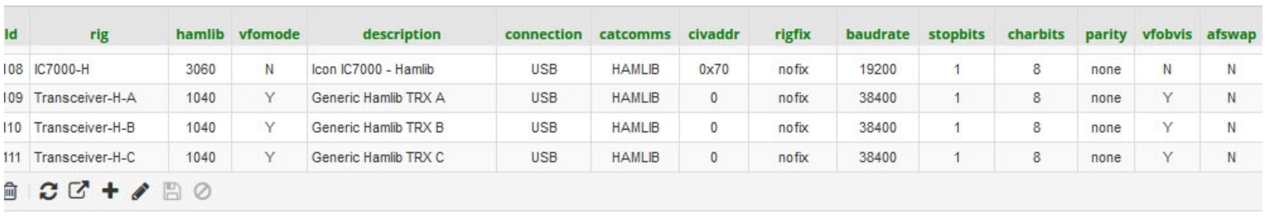
The Learning guide uses three example transceiver configurations using Hamlib.
These have a progressive build up of features and are in the supplied database as:
Transceiver-H-A, Transceiver-H-B, Transceiver-H-C. They use **--vfo mode**.

Transceiver-H-C is fairly well featured, but I attempt to restrict it to commonly used controls so that it can be switched fairly easily to any transceiver by simply changing the Hamlib reference number (and making any changes needed to the connection details) These configurations were developed on the FTdx101D but then worked fine on switching to the 20 yr old FT920.

I have also include a corresponding Hamlib progression that **does NOT use --vfo mode.**
Transceiver-H-A-NV, **Transceiver-H-B-NV** and **Transceiver-H-C-NV**.
These are currently working with my IC7000 using a CI-V connection (it doesn't have USB).

2.11 Learning guide - Transceiver-H-A (--vfo) Transceiver-H-A-NV (non --vfo)

This is a very limited Hamlib configuration.
Start piWebCAT in a web browser by entering the IP address 192.168.1.117 (or your changed value).
Click the Config button.
The configuration page always opens in the buttons editor (table: **buttonshl** for HAMLIB)
Select radio **Generic Hamlib TRX A** from the drop list selector. (Should be selected by default on new card).
Click the **radios** button on the button bar.



Id	rig	hamlib	vfomode	description	connection	catcomms	civaddr	rigfix	baudrate	stopbits	charbits	parity	vfobvis	afswap
108	IC7000-H	3060	N	Icon IC7000 - Hamlib	USB	HAMLIB	0x70	nofix	19200	1	8	none	N	N
109	Transceiver-H-A	1040	Y	Generic Hamlib TRX A	USB	HAMLIB	0	nofix	38400	1	8	none	Y	N
110	Transceiver-H-B	1040	Y	Generic Hamlib TRX B	USB	HAMLIB	0	nofix	38400	1	8	none	Y	N
111	Transceiver-H-C	1040	Y	Generic Hamlib TRX C	USB	HAMLIB	0	nofix	38400	1	8	none	Y	N

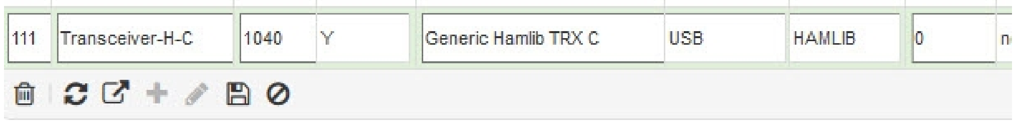
The **radios** table shows IC7000-H (H = HAMLIB) and Transceiver-H-A B and C. All four are Hamlib configured.
You are dealing with Transceiver-H-A now, but you might as well apply your settings to all three of A, B and C.

For each of A to C you need to enter:

- The radio's **Hamlib code** as determined above.
- **vfomode = Y** (selector edit) **Transceiver-H-A** uses **--vfo mode** which **rigctl** needs for a dual VFO system.
- All commands contain VFOA or VFOB (sometimes as a dummy)
  eg: `\set_level VFOA NR #` In piWebCAT the # is substituted with the value from the slider control.
  Transceiver-H-A-NV has **vfomode = N**. Commands are of the form: `\set_level NR #`
- **connection** = **USB** or **SERIAL** (Icom CI-V is SERIAL)
- **catcomms = HAMLIB** - to connect via the Hamlib rigctld system.
- **civaddr** if using an Icom rig. Enter in hex format eg: 0x70 as above for the IC7000.
- **rigfix = nofix**
- **baudrate** - This applies to USB connections as well as serial (I have to specify it in my FTdx101D)
- **stopbits = 1 or 2, charbits = 8, parity = none** for most radios.
- **vfobvis = Y** Whether VFOB to be displayed (maybe = N for single VFO radio)
- **afswap = N** initially. See section 1.6 Audio gain swapping

Editing a piWebCAT configuration table:

Editing is *inline* - simply click on the row for editing *and then click the* ***cell*** (otherwise you'll edit the id column)
Here, in the radios table, all fields except hamlib, description, civaddr and command are droplist selectors.



111	Transceiver-H-C	1040	Y	Generic Hamlib TRX C	USB	HAMLIB	0	n

Click the button to save your edit. (Hint: Hover the mouse over the buttons to reveal their functions)

Configuration editors

I am quite deliberately discussing the tables and their data on this minimal configuration of Transceiver-H-A ... in order to keep things simple !!

Note that all the database table records for a radio have the same **rig** field (eg: Transceiver-H-A)
and that this must also match the **rig** field of the radio's single record in the **radios** table.
If you misspell this field then that record is lost from view in piWebCAT. (You can retrieve with **MySQL Front.)**
Fortunately, when you add a new record to a table for a radio, piWebCAT automatically populates the **rig** field with the identifier of the current rig - so you don't get the chance to misspell it.

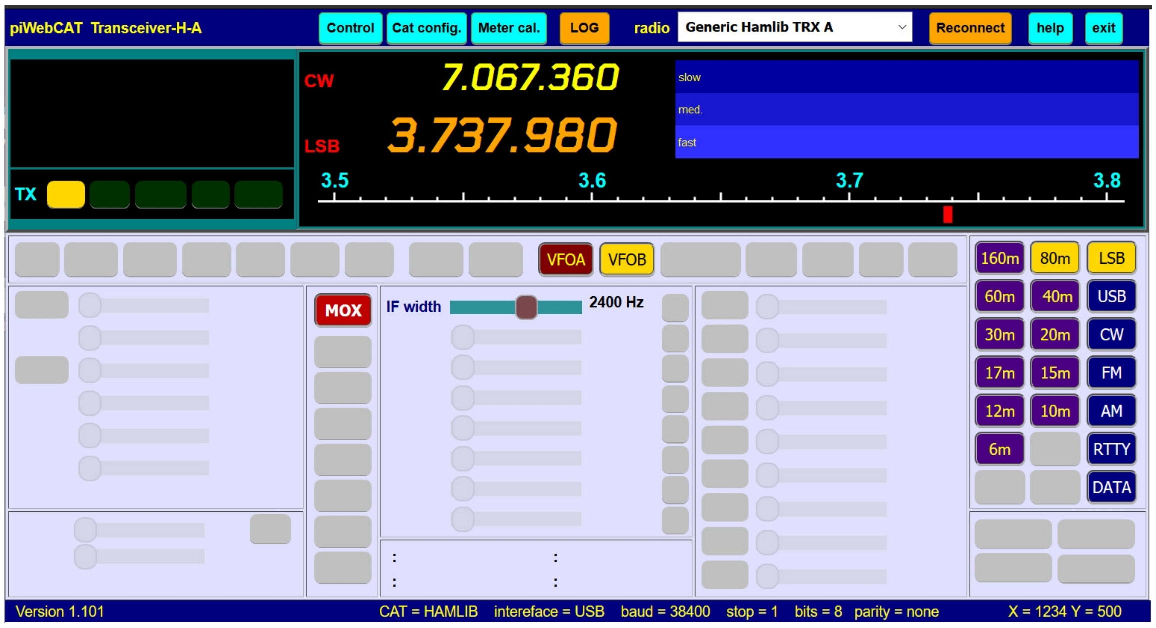
Now explore the other table edit buttons on the button bar (ie: buttonshl, carcodeshl etc)
These access the configuration tables and are restricted to the selected radios (which is very convenient!)
Have a good look at how the controls are configured.
Some import guidance with this:

- **buttonshl** table - This has data held by the **client.** (Extracted from server MySQL database at startup)
ie: btnno (unique id) button appearance, behaviour etc and the *data to send to the server*.

  The **code** field (eg: BAND) indicates the job.
vx is **A** or **B** if current-vfo specific, otherwise **X.**
The data for grouped buttons (eg: BAND, **action = G**) is always in fields **nset** and **nans**.
The data for other buttons is in fields **seton**, **setoff**, **anson** and **ansoff**
- **catcodehl** table - This has data used on the sever.
readmask and **setmask** are the read and write commands from server to Hamlib rigctld (or direct to the rig for non-Hamlib configuration).
The # character is substituted by the data from the client button configuration.
answermask with Hamlib is only used with rig commands (if no Hamlib alternative)
- **code = FREQ** In **catcodeshl** this is the frequency read and write command.
Most of the time, the calls from client to server **\set_freq** arise from user tuning actions.
Some arise from band switching or UP/DOWN buttons.
Calls to **\get_freq** are repetitive timer driven actions at intervals in the **timers** table.
- **Band switching** See **buttonshl** table. Many radios do not have a band switching CAT command, so we have to send a frequency in order to change band.
piWebCAT detects the frequency change and switches band on the display.
We send frequency here in this generic setup because it should work with most radios.
piWebCAT remembers the last frequency on each band and so will return to the last used frequency on *returning to a band*.
- **slidershl** table This table conifigures 25 sliders (and four text data items).
It contains both client and server data.
(Unlike buttons which have separate tables, ie: button client data in table **buttonshl** and server data in table **catcodeshl**.)
Server data is similar to **catcodeshl** but with numeric data: eg **\set_level VFOA NR #**
Client fields **min** and **max** define the range of data as read / sent to / from the radio.
piWebCAT matches the **min** and **max** value to the full range of slider travel.
Fields **mult, divide**, **offset** and **decpoint** scale the numeric data for presentation to the right of the slider. Field **units** is optional units display.
Field **lookup** (Y or N) determines whether a lookup table should be used when the relationship between incoming data and display value is non-linear,
eg: 1 2 3 4 displaying as 50Hz 100Hz 200Hz 400hz.
- **timings** table This defines the repeat intervals at which different repetitive tasks are queued on the client for transmission to the server.
The shorter **readqueue** interval is the polling rate for sending from the queue.
Note that the use of this managed queue was vital in maximising use of the client <> server data bandwidth.

Now explore these table editor buttons selecting with the button bar (ie: buttonshl, carcodeshl etc)
The editors work on data filtered to present only the selected radio (which is very convenient!)

Now, with the radio switched on, click the top bar Control button.
piWebCAT should link to your radio and display the VFO frequencies and modes.
Band and mode switching should be operational.



Version 1.101 CAT = HAMLIB intereface = USB baud = 38400 stop = 1 bits = 8 parity = none X = 1234 Y = 500

Note that you have to wait 2 seconds or so after band change for the display to stabilise before you can tune and operate other controls. This interval is longer with many controls configured.
If you switch bands simply by changing VFO, then once you have visited each band, the stabilisation is much faster. This is because piWebCAT remembers the complete last control set for each VFO.

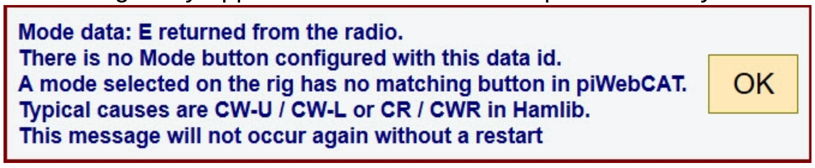
Note that **IF width** is on this minimal display.
This is because Hamlib reads and writes IF width and band in a single command.
This gives piWebCAT coding some work to do in correctly processing the dual data returns.

Mode and IF Width share common commands
Note that Hamlib combines mode and IF Width into single \get_mode and \set_mode commands.
This adds some complexity, as I have to separate and combine the two values.
The detailed explanation of this is in Section: 8.12 rigctl mode and bandwidth

Note the **MOX** button. It should be able to perform T/R switching.
It should respond to T/R switching from the radio.
I include it at this stage because Tx / Rx status is tested after band change.
Mode button group

MODE button and button groups caution.

My FTdx101D has fifteen modes.
piWebCAT, here has seven mode buttons configured as a group linked to seven of the fifteen modes.
If I do all my mode selection on piWebCAT, then there is no problem.
If, on the rig, I select a mode that has no corresponding piWebCAT button, then the message box below will appear as piWebCAT starts or when the such a mode is selected.
The message only appears once and then is supressed until you restart piWebCAT.



Mode data: E returned from the radio.
There is no Mode button configured with this data id.
A mode selected on the rig has no matching button in piWebCAT.
Typical causes are CW-U / CW-L or CR / CWR in Hamlib.
This message will not occur again without a restart

OK

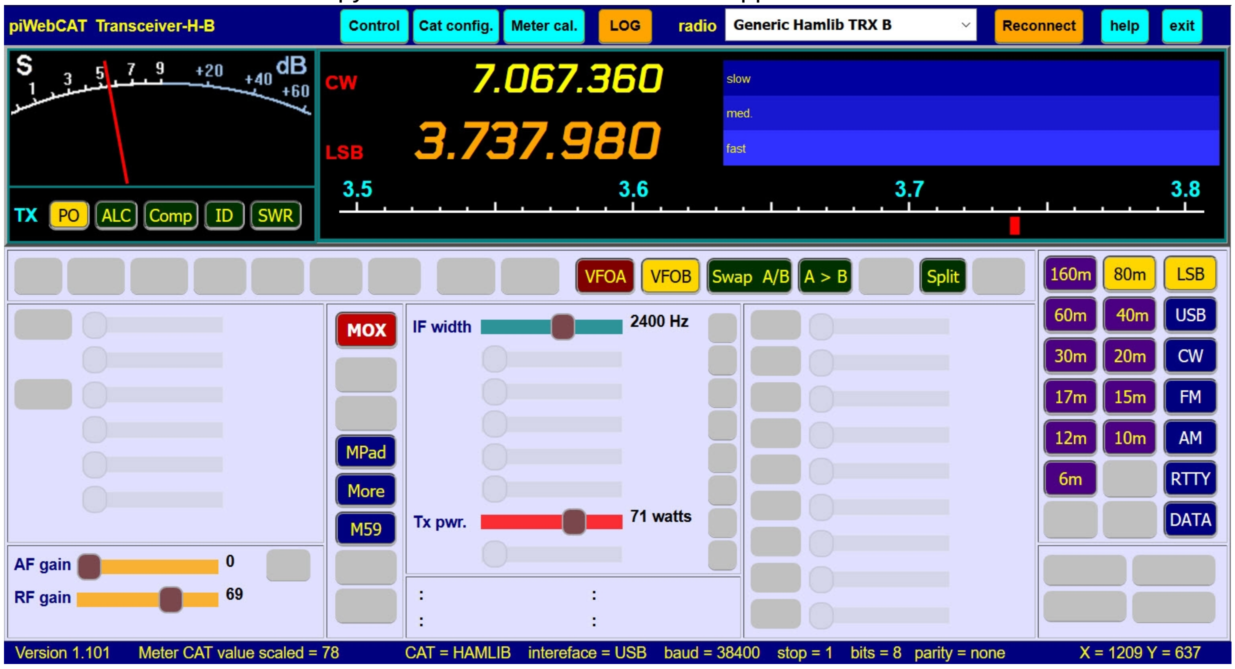
A similar generic message box will appear if the problem arises in other button groups.
The box occurs on start up. The box will appear if the unrecognised condition is subsequently selected but only if the button group has **active=S** which is sync mode for regular updates.

See also end of section 3.7 Button group problem

2.12 Learning guide -Transceiver-H-B (--vfo) Transceiver-H-A-NV (non --vfo)

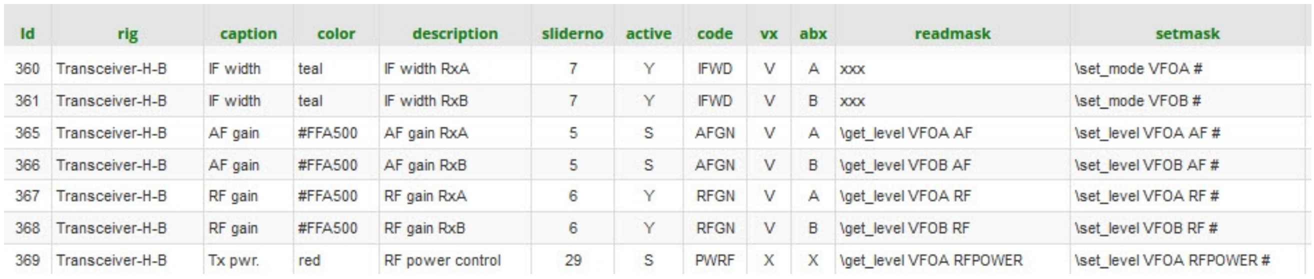
- metering and a few controls added

Now change the radio selector to **Generic Hamlib TRX B**
This adds Rx and Tx metering, VFO swap, copy and split, Tx power and AF and RF gain.
Note that there is no VFO copy B > A button. Hamlib doesn't support it.



Version 1.101 Meter CAT value scaled = 78 CAT = HAMLIB intereface = USB baud = 38400 stop = 1 bits = 8 parity = none X = 1209 Y = 637

Examine the **slidershl** table. The left side is shown below



Id	rig	caption	color	description	sliderno	active	code	vx	abx	readmask	setmask
360	Transceiver-H-B	IF width	teal	IF width RxA	7	Y	IFWD	V	A	xxx	\set_mode VFOA #
361	Transceiver-H-B	IF width	teal	IF width RxB	7	Y	IFWD	V	B	xxx	\set_mode VFOB #
365	Transceiver-H-B	AF gain	#FFA500	AF gain RxA	5	S	AFGN	V	A	\get_level VFOA AF	\set_level VFOA AF #
366	Transceiver-H-B	AF gain	#FFA500	AF gain RxB	5	S	AFGN	V	B	\get_level VFOB AF	\set_level VFOB AF #
367	Transceiver-H-B	RF gain	#FFA500	RF gain RxA	6	Y	RFGN	V	A	\get_level VFOA RF	\set_level VFOA RF #
368	Transceiver-H-B	RF gain	#FFA500	RF gain RxB	6	Y	RFGN	V	B	\get_level VFOB RF	\set_level VFOB RF #
369	Transceiver-H-B	Tx pwr.	red	RF power control	29	S	PWRF	X	X	\get_level VFOA RFPOWER	\set_level VFOA RFPOWER #

Note that **AFGN** and **RFGN** each have two records, one with **abx = A** for VFOA and one with **abx = B** for VFOB.
These two entries are used on the ***server***.
The ***client*** (web browser) data is extracted from only one of these two records
... the first to be encountered in the startup. So for safety, we put the client data in both, ie: **vx**, **mult**, **div** etc)
Both client and server data will have the **code = AFGN** or **code = RFGN** to link to the server records.
The client records have **vx = V** (for VFO).
This makes the client send gain change commands to the server with the current VFO selection, A or B.
The server then knows whether to communicate with the rig using the A or B record for gain setting or reading.
For a more detailed explanation of this, see 3.10 README vx and abx

active = S (rather than active = Y) adds the slider to a list of controls for regular update to follow any changes on the rig. (Doing this with all controls it might use too much data transfer bandwidth.)

The **sliderno** field contains the slider's fixed reference number. See 3.6 Button and Slider numbering

Note that Tx power (**PWRF**) only has a single record in this table **slidershl**.
This is because it is not specific to VFOA or VFOB.
The client data has **vx=X**. The server data has **abx=X**. The command from client contains X (rather than A or B).
Note the Hamlib commands in:

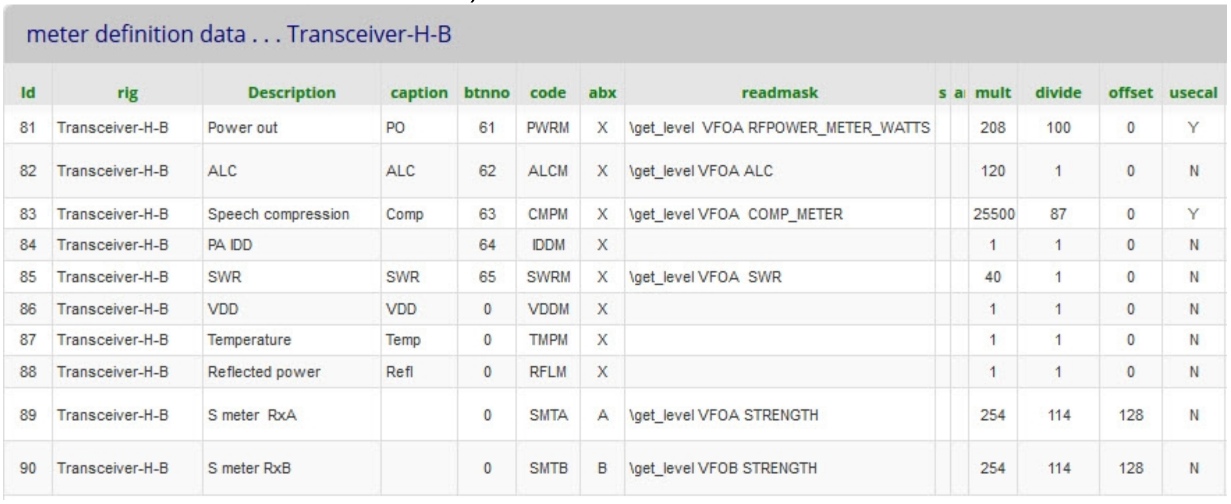
- **readmask**: `\get_level VFOA RFPOWER` (or `\get_level RFPOWER` in non---vfo mode)
- **setmask**: `\set_level VFOA RFPOWER #` The # character is replaced by the data from the client.

In **--vfo mode**, the VFOA parameter is a dummy. The RF power control is not VFO specific, but Hamlib rigctld requires us to include either VFOA or VFOB.... because we are in Hamlib VFO mode
(As configured for the radio in the **radios** table.)

Metering - S meter Tx meters

The Tx meter buttons determine which transmit meter will be displayed on transmit.
They can be switched whilst on receive or whilst on transmit.
They cause the appropriate meter background bitmap image to be presented on transmit and the appropriate value for display to be repetitively read from the radio. See Section 9.1 Meter background images
There are five transmit bitmaps and one for receive (the S meter).The bitmaps are accessible for editing (care!).
The meter scales can be made to accurately match those on the radio by optional use of the **metercal** table.
The **metercal** table contains up to 20 user-defined calibration points per meter with linear interpolation.

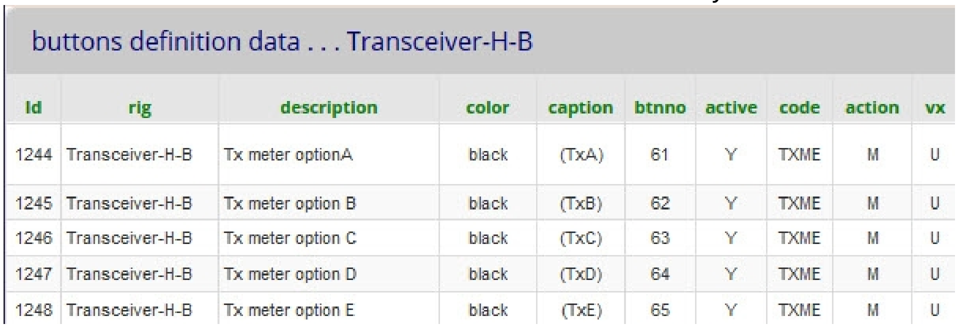
The image below is of the meter table for Transceiver-H_B with unused columns collapsed for a compact image.
Note that only five of the eight Tx metering records are configured. These have **btnno** field = 61 to 65.
The unused records have btnno field = 0)



meter definition data . . . Transceiver-H-B

Id	rig	Description	caption	btnno	code	abx	readmask	s	a	mult	divide	offset	usecal
81	Transceiver-H-B	Power out	PO	61	PWRM	X	\get_level VFOA RFPOWER_METER_WATTS			208	100	0	Y
82	Transceiver-H-B	ALC	ALC	62	ALCM	X	\get_level VFOA ALC			120	1	0	N
83	Transceiver-H-B	Speech compression	Comp	63	CMPM	X	\get_level VFOA COMP_METER			25500	87	0	Y
84	Transceiver-H-B	PA IDD		64	IDDM	X				1	1	0	N
85	Transceiver-H-B	SWR	SWR	65	SWRM	X	\get_level VFOA SWR			40	1	0	N
86	Transceiver-H-B	VDD	VDD	0	VDDM	X				1	1	0	N
87	Transceiver-H-B	Temperature	Temp	0	TMPM	X				1	1	0	N
88	Transceiver-H-B	Reflected power	Refl	0	RFLM	X				1	1	0	N
89	Transceiver-H-B	S meter RxA		0	SMTA	A	\get_level VFOA STRENGTH			254	114	128	N
90	Transceiver-H-B	S meter RxB		0	SMTB	B	\get_level VFOB STRENGTH			254	114	128	N

Below are the five Tx metering button configurations in table **buttonshl**.
None of the data fields are used for these buttons and so they are omitted here for a more compact image.



buttons definition data . . . Transceiver-H-B

Id	rig	description	color	caption	btnno	active	code	action	vx
1244	Transceiver-H-B	Tx meter optionA	black	(TxA)	61	Y	TXME	M	U
1245	Transceiver-H-B	Tx meter option B	black	(TxB)	62	Y	TXME	M	U
1246	Transceiver-H-B	Tx meter option C	black	(TxC)	63	Y	TXME	M	U
1247	Transceiver-H-B	Tx meter option D	black	(TxD)	64	Y	TXME	M	U
1248	Transceiver-H-B	Tx meter option E	black	(TxE)	65	Y	TXME	M	U

The Tx meter buttons work as a group but have action = M rather than the usual G for groups.
They have fixed positions beneath the meter and have fixed **btnno** fields = **61 to 65**.
They link, using the **btnno** to five of the records in the **meterhl** table shown above.

The meter captions are defined in the **meterhl** table (The caption fields in **buttonshl** are not used)
The CAT commands are shown above, eg: \get_level VFOA RFPOWER_METER_WATTS
(VFOA is a dummy parameter because we are running rigctl in --vfo mode)

The power meter above is configured for my FTdx101D.
Initally, with no scaling (mult=1, div=1) it was reading less than half scale at 100w.
Setting mult=208 and div=100 set the 100w point correctly but lower power levels were very inaccurate.
So I performed a calibration (examine **metercal**) and then switched it in by setting **meterhl.usecal = Y.**
The result was near perfect match between radio and piWebCAT. See Section 9.2 Meter calibration

The S meter here doesn't need the calibration table as it was carefully modelled on that of the FTdx101D.
The calibration table is available when needed .... as it was for the IC7000 configurations.

2.13 Learning guide - fixed controls - button actions

Buttons controls have a **code** field.eg: KEYR, BAND, MODE, SPSW.

- The **code** field appears in the **buttonshl** table (which is data held in the client)
- The **code** field appears in the **catcodeshl** table (which is data used on the server.)
- The **code** field provides a linking identifier for commands sent from client to server.
- Likewise, **sliders** use a code field in the same way, but as discussed earlier, for sliders, both client and server data are configured in the single **slidershl** table.

All **code** field values should have meaningful names.
For most of them, you are free to choose your own names, eg: KYER instead of KEYR etc.
Clearly, it is essential that the corresponding **code** fields in **buttonsh**l and **catcodeshl** are the same!

However, we have a system here in which configuration data can be shared between users, and so perhaps there is a case for some standardisation, eg: my codes!

A layout with button and slider numbering is shown in section 3.6 - Button and Slider numbering

Fixed buttons

Some buttons use fixed codes which you must not change.
A few buttons are fixed in postion.
Some buttons are in positions which you could change ... but why? eg: BAND and MODE.

Buttons with fixed location and function

These buttons have fixed locations (btnno values) and fixed code fields:

- VFOA, VFOB, Swap A/B and MOX (**btnno** = 16, 17, 79 and 59)
  code fields are fixed at VFO, VFO, SWAP and MOX.
- Five Tx metering buttons beneath the meter (**btnno** = 61, 62, 63 ,64 and 65)
  They all have **code = TXME** and **action = M**
- Eight slider reset-to-default buttons (**btnno** = 51, 52, 53, 54, 55, 56, 89, and 90)
  These have **action = R** and no code field. We don't configure these other than to set as active/inactive
  This is because the buttons only action is via internal links to hard coded functions in piWebCAT code.

Note that the Tx metering and slider reset buttons operate as so called 'radio buttons' ie: in a group where only one can be selected.
Elsewhere, group behaviour is specified by **action = G** and a shared code field, eg: BAND, MODE, ATTN.

Buttons with a fixed code but free location.

These can be moved by simply changing their **btnno** field in table **buttonshl**.
All my BAND and MODE buttons are in the right hand panel with buttons for 14 bands and 8 modes.
Users will hopefully not wish to move these.
An exception to this is that on an HF radio, the 4m, 2m and 70cm positions could be used for extra modes.
The buttons with fixed code are:

BAND	Band switching grouped buttons
MODE	Mode switching - grouped buttons..
MPAD	launch popup memory keypad (no server action)
MORE	launch popup window with 24 extra buttons (no server action)
RPGO	CW repeat toggle on/ff - a piWebCAT repeat

Sliders with a fixed code but free location.

Sliders with fixed code fields that can be located where you choose have code fields:

AFGN	AF gain - fixed because it is linked to internal AFgain swapping code.
IFWD	IF width -fixed because it is linked in Hamlib configurations to MODE

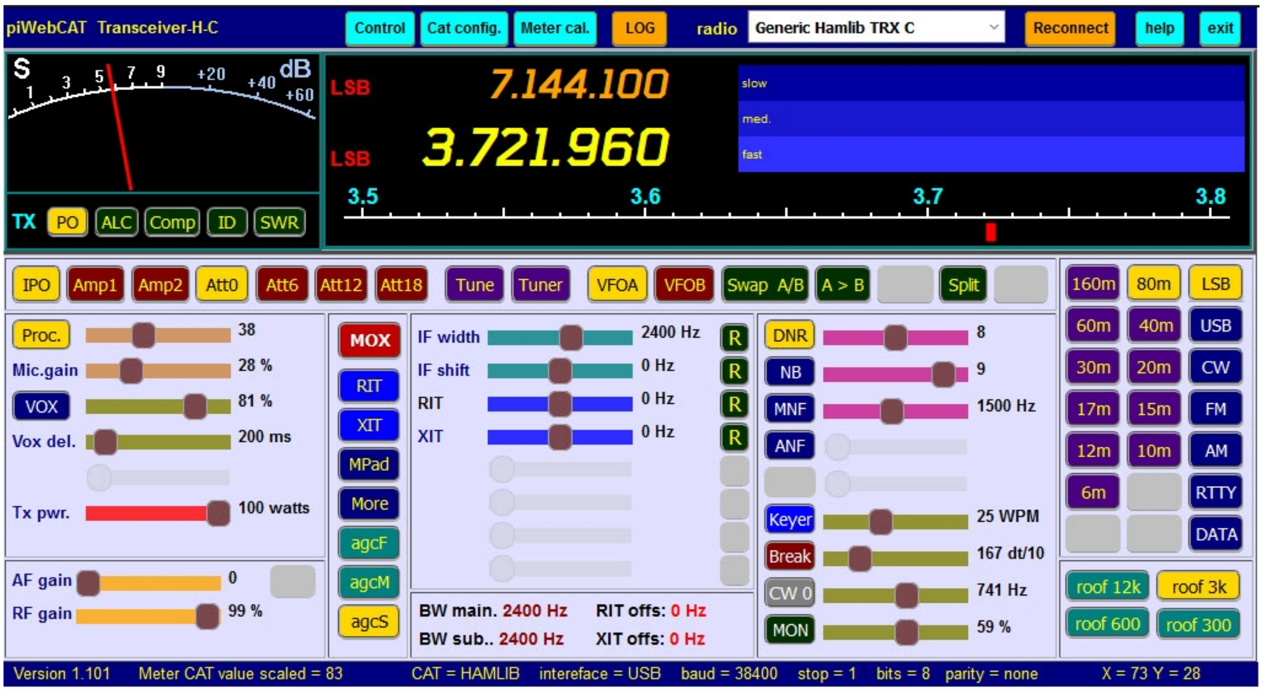
2.14 Learning guide Transceiver-H-C ---vfo mode

Select **Transceiver-H-C** - the piWebCAT window is shown below.

19 new buttons and 14 new sliders have been added to the previously studied Transceiver-H-B layout.
All of these except CW zero use Hamlib **rigctl**. (CW zero is not supported and needs a direct rig command.)

Note that piWebCAT was developed with the FTdx101D whilst keeping in mind the need to operate with many different radios. So, for example, I have four roofer buttons. But you could use them for some other function.

- Tx Power slider has moved. This move is a simple change of slider number in table **slidershl**.
- Proc, Mic-gain, VOX and VOX delay sliders added. Two have adjacent buttons which double as labels. These four are commonly used controls and I would expect that most users would keep these positions.
- The middle panel sliders have adjacent **R** buttons = reset to default.
  The right panel sliders have adjacent on/off buttons which double as labels. (Set slider captions to 'nocap')
  IF shift is in the middle panel because it needs reset to zero.
  I have positioned RIT and XIT in the middle panel because reset to default (zero Hz) is essential.
  But RIT and XIT on/off buttons are also needed. They are almost adjacent in the narrow 8-button panel !!!
- Roofer buttons are bottom right. These are not dedicated buttons - they could be used for something else. Roofer buttons could be elsewhere if you wish.
- The three CW buttons and sliders are in the right hand panel. I copied them (using MySQL Front) from my FTdx101D-H configuration and then moved them from the MORE buttons popup to here, simply by changing the buttons' numbers in the **buttonshl** table **btnno** field.
- Note the four **text** items middle bottom. These are configured in the slidershl table.
  Their configuration was taken directly from the FTdx101D-H configuration.
  They are discussed in detail in section 2.4 Text display box.
- Note that there there is no VFO B > A copying button. This function is not supported by Hamlib.
  I could, for the FTdx101D implement it by using \send_cmd_rx BA; 0 (a direct rig command).
- I have added three AGC buttons: fast, medium and slow. Examination of the **buttonshl** table reveals data values (**nset** and **nans**): fast=2. medium=5 and slow=3.
  The values in theFdx101D CAT manual are 1, 2 and 3. So where do we find 2, 5 and 3 ?
  See section 2.17 Learning: Hamlib data? also section 8.5 rigctl - at the command line



BW main. 2400 Hz RIT offs: 0 Hz
BW sub.. 2400 Hz XIT offs: 0 Hz

Version 1.101 Meter CAT value scaled = 83 CAT = HAMLIB intereface = USB baud = 38400 stop = 1 bits = 8 parity = none X = 73 Y = 28

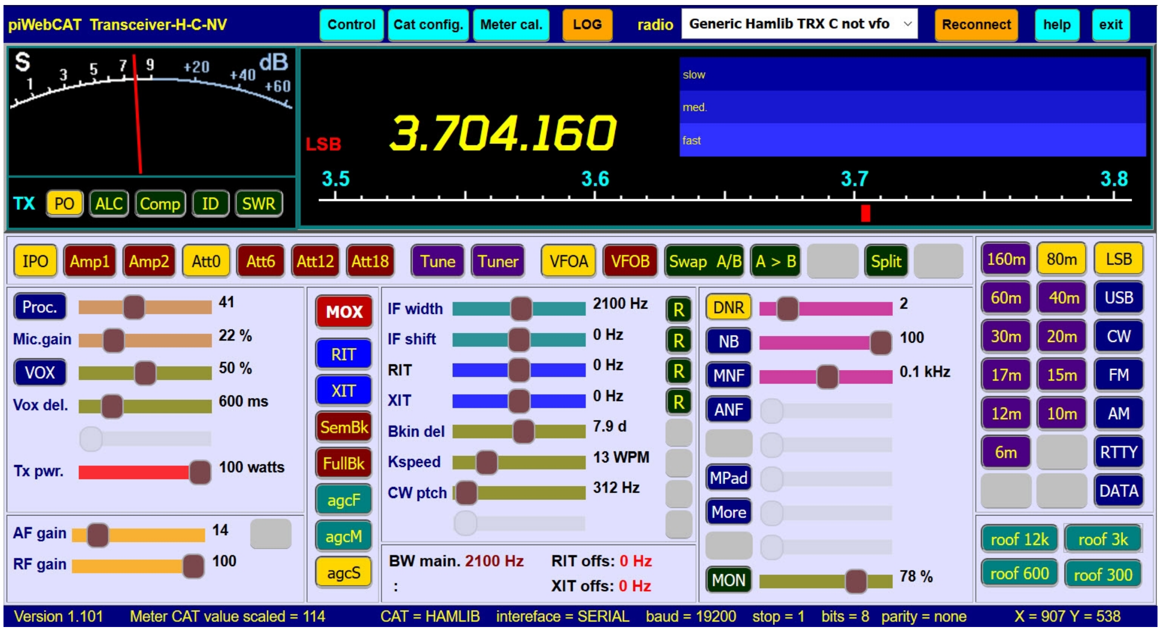
2.15 Learning guide Transceiver-H-C-NV non ---vfo mode

Select **Transceiver-H-C** - the piWebCAT window is shown below.
26 new buttons and 14 new sliders have been added to the previously studied Transceiver-H-B layout.
Some buttons have been move. This done very easily by changing the buttons numbers.
(A button and slider numbering layout is shown in section 3.6 Button and Slider numbering).
All of these controls use Hamlib **rigctl**

Some controls are ineffective in my IC7000, eg: Roofer buttons, RIT, XIT, Tune, Tuner, IF Shift.
These have been disabled in the configuration on the SD card and therefore do not appear. See next page.
They are easily reinstated by changing their active field from **N** to **Y** or **S** in **the buttonshl** or **slidershl** table.

I suggest enabling new controls with **active=Y**. They will then only be read once.
If you enable with **active=S** (sync), then piWebCAT attempts to read the parameter from the rig every few seconds and you have succession of error messages. Switch to **active=S** when your a happy that they are ok.

They are all read at startup. If any one is not supported there will a brief delay and '? disconnection' error box.
It can be tricky to identify which item is doing this (either through not being supported or a mis-spelt command)
It is therefore advisable to only re-enable one control at a time.



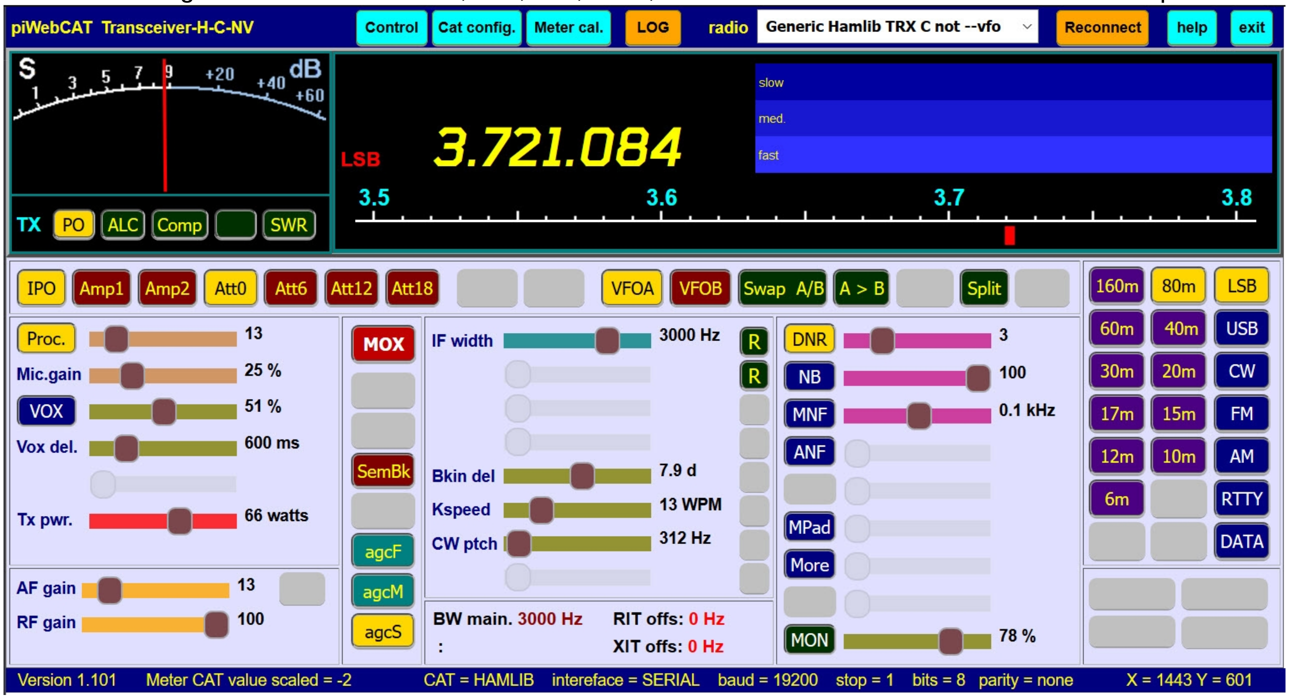
Version 1.101 Meter CAT value scaled = 114 CAT = HAMLIB intereface = SERIAL baud = 19200 stop = 1 bits = 8 parity = none X = 907 Y = 538

- Proc, Mic-gain, VOX and VOX delay sliders added. Two have adjacent buttons which double as labels.
  These four are commonly used controls and I would expect that most users would keep these positions.
- The middle panel sliders have adjacent **R** buttons = reset to default.
  The right panel sliders have adjacent on/off buttons which double as labels. (Set slider captions to 'nocap')
  IF shift is in the middle panel because it needs reset to zero.
  I have positioned RIT and XIT in the middle panel because reset to default (zero Hz) is essential.
  But RIT and XIT on/off buttons are also needed. They are almost adjacent in the narrow 8-button panel !!!
- Roofer buttons are bottom right. These are not dedicated buttons - they could be used for something else.
  Roofer buttons could be elsewhere if you wished. (Not used in IC7000)
- Note the three CW sliders are in the middle panel. I positioned them here near the break-in buttons.
  They could easily be move elsewhere by changing their **sliderno** fields in table slidershl.
  The MORE and MPad popup buttons were moved by changing the **btnno** field in the **buttonshl** table.
- Note the four **text** items middle bottom. These are configured in the slidershl table.
  They are discussed in detail in section 2.4 Text display box.
- Note that there is no VFO B > A copying button. This function is not supported by Hamlib.
  I could, for the IC7000 implement it by using \send_cmd_rx BA; 0 (a direct rig command).
- I have added three AGC buttons: fast, medium and slow. Examination of the **buttonshl** table
  reveals data values (**nset** and **nans**): fast=2. medium=5 and slow=3.
  The values in the IC7000 CAT manual are 1, 2 and 3. So where do we find 2, 5 and 3 ?

See section 2.17 Learning: Hamlib data? also section 8.5 rigctl - at the command line

Examine the configuration of the Breakin delay, Keyer speed and CW pitch sliders in table **slidershl**.
My intention is usually to display in the same units as on the rig.
Eg; Bkin delay: the values on the rig are **2.0d to `12.8d.** The data received from rigctld data is 1 to 255.
The data range must be made to correspond to the full range of the slider. The text presentation
to the right of the slider must be **2.0d to `12.8d.** piWebCAT's formatting fields in table **slidershl** provide the means to achieve this. The process will then operate in both directions, ie: slider controlling the rig parameter and slider following the parameter's value on the rig (if **active=S** = sync is set).

The same configuration but with Roofer, RIT, XIT, Tune, Tuner and IF Shift inactivated for IC7000 operation.



Proc. 13
Mic.gain 25 %
VOX 51 %
Vox del. 600 ms
Tx pwr. 66 watts
AF gain 13
RF gain 100
IF width 3000 Hz
Bkin del 7.9 d
Kspeed 13 WPM
CW ptch 312 Hz
DNR 3
NB 100
MNF 0.1 kHz
MON 78 %
BW main. 3000 Hz :
RIT offs: 0 Hz
XIT offs: 0 Hz

Version 1.101 Meter CAT value scaled = -2 CAT = HAMLIB intereface = SERIAL baud = 19200 stop = 1 bits = 8 parity = none X = 1443 Y = 601

2.16 Learning guide: Buttons groups (and more data values than buttons?)

Grouped buttons

Only one button of a group can remain selected and its selection deselects the others in the group.
Buttons are defined in table: **buttonshl** (HAMLB), **buttons** (ASCII and YAESU5) and **buttonsciv** (CIV).
The button table data is used on the client (ie: in javascript code in the web browser)
piWebCAT recoginises buttons as belonging to a group if they have **action = G** and the same **code**.

The group acts on a single rig parameter and each button is associated with a different value for that parameter. eg: three buttons controlling AGC decay: fast, medium and slow.
Note that in **dual receiver systems** when using **--vfo mode**, the group may have **two server records** in the **catcodes** table (or **catcoedshl** or **catcodesciv**). One record is for VFOA and one is for VFOB.

The data values for grouped buttons are contained in fields **nset** and **nans** (or **bgsdata** for Icom CIV).
The **nset, nans** and **bgsdata** fields are only used for grouped buttons.
(Toggling and single shot buttons use **anson**, **an**soff, **seton** and **setoff**)

- The **nset** field specifies the data value to be sent to the server when the button is clicked.
- The **nans** field is used for data reads from the server (ie: from the rig). piWebCAT scans the group for a button with an **nans** value that matches the returned data and sets it. The others are cleared.

Buttons states are set from the rig values at startup.
Button state are subsequently repetitively updated only if the buttons are configured with **active=S** (sync).
(It is wise to avoid having too many buttons on auto-update with active=S .... *but be aware that a group of buttons only needs a single update*!)

It is possible to have more values for the controlled parameter than there are buttons to match.
For example: **AGC decay on the FTdx101D.**
We can configure **fast**, **mid** and **slow** decay times (for each mode) in the rig's menus and then select **fast**, **mid** or **slow** either on the rig touch screen or via CAT control.

From the CAT manual (GT command)

- **Setting ACG** (**nset** values). There are five options: agc off, fast, mid, slow or auto.
- **Reading AGC** (**nans** matches).There seven options: agc off, fast, mid, slow, auto-fast, auto-mid, auto-slow.

The question immediately arises as to what to do with seven data values returning to fewer buttons.
Firstly, I choose not to deal with AGC OFF (we can do that on the rig when we really need to).
There are two ways to deal with this:

1. Provide **four buttons**: FAST, MID, SLOW and AUTO. (The rig's user interface does this)
   Configure **nset** values to send: fast, mid, slow and auto.
   Configure **nans** values so that: fast sets FAST, mid sets MID, slow sets SLOW
   and auto-fast, auto-mid and auto-slow all set AUTO.
2. Provide **three buttons**, FAST, MID and SLOW . (So I can't then actually set auto)
   Configure **nset** values to send: fast, mid and slow.
   Conigure **nans** value so that: fast and auto-fast sets FAST, mid and auto-mid sets MID,
   and slow and auto-slow sets SLOW.

For the direct (not Hamlib) FTdx101D configuration, I use option 1.
For FTdx101D-H I have no choice other than to use option 1. This is because Hamlib rigctld returns fast, mid, slow or auto. ie: it combines auto-fast, auto-mid and auto-slow into a single auto value before returning the data to piWebCAT.

Combining data values (only relevant to grouped buttons)
Examine the FTdx101D configuration buttons table. Find the **four AGC decay buttons**.
The **nans** field for the ACGauto buttons is set to **4 | 5 | 6**.
The **|** character means OR. 4,5 and 6 are the return codes for auto-fast, auto-mid and auto-slow.
Thus any auto-ACG return value will select the auto button.

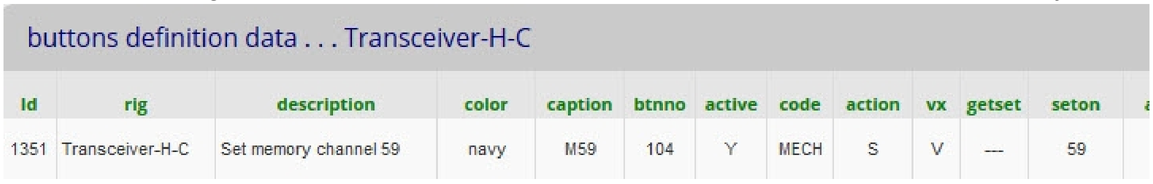
Examine he FTdx101D-H configuration buttonshl table.
You will see returns CW and CWR combined on the CW button using **CW | CWR**. Likewise **RTTY | RTTYR**.

2.17 Learning guide - Memories Mem keypad More buttons

Transceiver memories

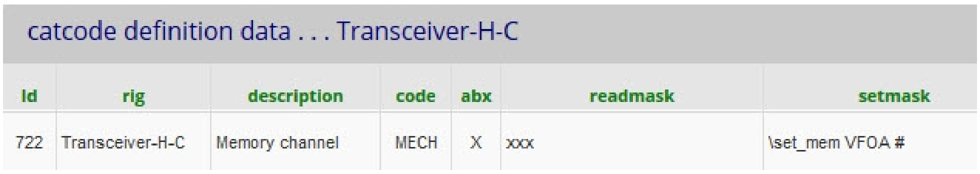
Transceiver memories usually contain frequency and mode parameters and possibly other parameters.
The memories must be set up on the transceiver. piWebCAT cannot do this.
piWebCAT simply launches a memory by its number.
If there are memory banks (eg: A, B, C) then these must be preselected on the transceiver.

Below is the configuration data in **buttonshl** and **catcodeshl** for a button to launch memory 59.



buttons definition data . . . Transceiver-H-C

Id	rig	description	color	caption	btnno	active	code	action	vx	getset	seton
1351	Transceiver-H-C	Set memory channel 59	navy	M59	104	Y	MECH	S	V	---	59



catcode definition data . . . Transceiver-H-C

Id	rig	description	code	abx	readmask	setmask
722	Transceiver-H-C	Memory channel	MECH	X	xxx	\set_mem VFOA #

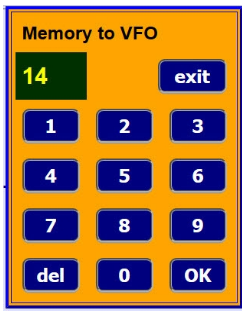
The button must use **code=MECH**. The Hamlib command is \set_mem VFOA # .
The server substitutes the # with value 59 from the client command message.
You might label the button as mem59 OR 3.72 (ie: the frequency)
You can set up as many MECH buttons as you wish .... but perhaps the memory keypad is more efficient.

MTOV command - important

Some transceivers use a memory channel command that selects the memory and applies it to the VFO
Some transceivers require two commands: Select memory and then Apply to VFO.
Some transceivers have all three commands thus giving both the above options.

Hamlib's **set_mem VFOA 59** selects the memory. This command is configured for MECH as above
It has to be followed by **\vfo_op VFOA TO_VFO** This applies the selected memory to the VFO.
The \vfo_op command is configured only in catcodedshl. It has code = MTOV (move to VFO)
You could configure a button to make catcodeshl send out \vfo_op VFOA TO_VFO, but you don't need to.
piWebCAT, internally sends a MTOV command to the server 1 second after every MECH command.
If you are not using Hamlib and you don't need MTOV, then simply do not configure it in catcodes.

Memory keypad

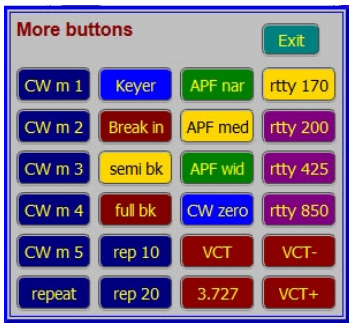


Examine the MPad configuration in **buttonshl**.
It must have **code=MPAD** and **action=S** (Single action button)

The Mpad button has no action other than to display the keypad.
The user selects a memory and click OK.

The OK button sends an MECH command to the server
... exactly the same as a dedicated MECH button would do.

More buttons keypad - example below from my FTdx101D-H configuration



This provides 24 buttons in four columns with:
btnno = 111 to 116, 121 to 126, 131 to 136 and141 to 146.
It is launched from a button with:
caption = More and **code=MORE** **action = S** (single)

The More button can be positioned in any free button position.
You choose the colour and caption like any other buttons.
The More button has no action on server / transceiver.

Button configuration is the same as the buttons on the main window.

2.18 Learning guide - finding Hamlib data values

This section describe how to find Hamlib control value using **rigctl** at the command line.
I use the example of the three ACG buttons which were added in Transceiver-H-C.
For more extensive information on Hamlib data values, see Section 8.5 rigctl - at the command line

AGC buttons

These were actually configured for the FTdx101D.
The actual control values may or may not be correct for your radio.

*Examine table **buttonshl**.*
Find the records for ACG **fast**, **medium** and **slow** buttons.
The correct data values in **nset** (setting AGC delay) and **nget** (reading AGC delay) are **2**, **5** and **3**.

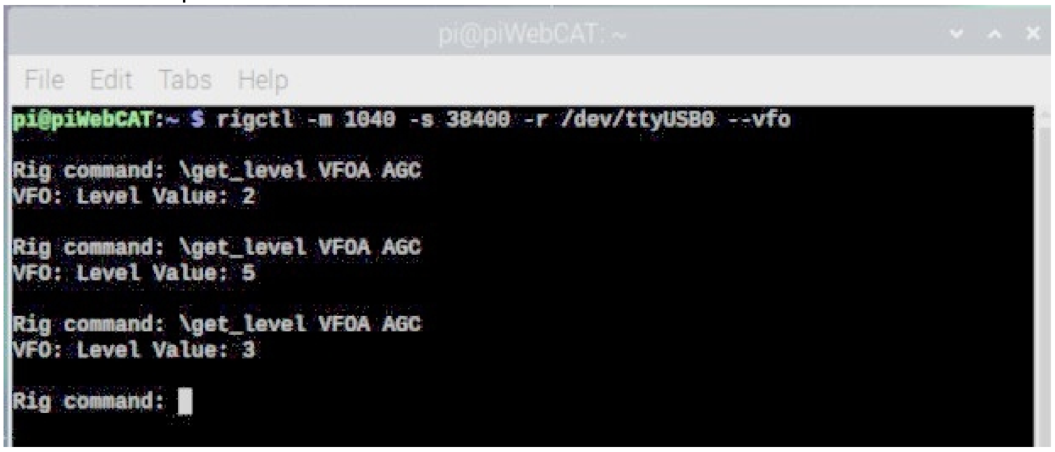
Where did I find these?? The following explains how.

- The FTdx101D CAT manual shows the GT command with values **fast = 1**, **med = 2** and **slow = 3**.
  (The actual decay times of fast, medium and slow are menu settings and differ between modes.)
- The Hamlib commands are **\get_level VFOA AGC** and **\set_level VFOA AGC #** where # is the value.
  However I couldn't find any information on the actual Hamlib control values.

So where do I find the unexpected values of **2**, ***5*** and **3** to configure in piWebCAT.?

The answer is to use **rigctl** at the Linux command line.
This is extremely useful for testing and in particular for checking the ranges of returned data.
eg: some command return 0.000 to 1.000, some return 0 to 255, some return in Hz.

The screen dump below is from the RPi Linux Terminal window.



```
pi@piWebCAT:~ $ rigctl -m 1040 -s 38400 -r /dev/ttyUSB0 --vfo

Rig command: \get_level VFOA AGC
VFO: Level Value: 2

Rig command: \get_level VFOA AGC
VFO: Level Value: 5

Rig command: \get_level VFOA AGC
VFO: Level Value: 3

Rig command: █
```

Here, I started **rigctl** for the FTdx101D (1040) at 38400 baud and for USB connection.
I set the rig to **fast** AGC and the entered **\get_level VFOA AGC**
I then repeated this for **medium** and **slow** AGC.

From the above window you can see that the returned levels were **2**, **5** and **3**.
These were used in the piWebCAT Hamlib configuration.

Starting rigctl in the RPi terminal for non --vfo mode and for --vfo mode

`rigctl -m 3060 -s 19200 -r /dev/ttyAMA0` for **non --vfo mode** and
`rigctl -m 3060 -s 19200 -r /dev/ttyAMA0` for **--vfo mode**
(/dev/ttyAMA0 is a serial connection. For USB, we specify .../dev/ttyUSB0)

piWebCAT, internally uses similar start up commands for **rigctld**

*Setting **vfomode = Y** in the database **rigs** table invokes --vfo mode - which must thereafter be used in all Hamlib commands.*

2.19 Using web-browser diagnostics to find configuration errors.

For more examples, see also section 14.3 - Development tools

Web Browser Inspect element facility

This is a right mouse option on most web browsers.
It is particularly good on **Firefox developer** and **Chrome.**

It provides a huge amount of information on traffic to and from websites.
It has very useful debugging facilities.

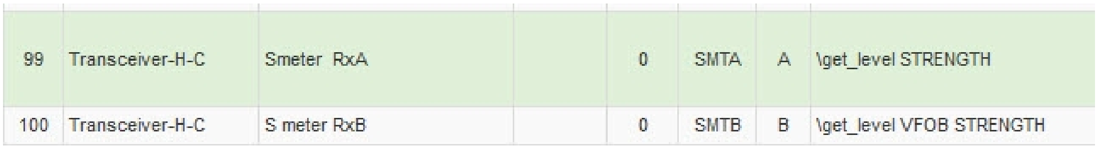
Its most useful application is in examining Ajax requests to the RPi server and the resulting responses.

Additionally, if there is an error in the server PHP code which is processing the request,
the PHP system echos back a detailed error message containing the location of the problem.
If you can't interpret it, then you can always send a copy to me / user group.

Example of incorrect Smeter configuration (as in the video!)

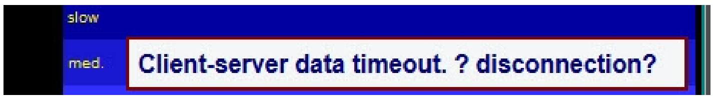
Problem simulation: Illegal command syntax - A common problem during configuration.
Using **Transceiver-H-C** with FTdx101D:
I deliberately introduce an error into table **meterhl:**

- I omitted the VFOA parameter from S meter RxA **readmask**
  ie: `\get_level STRENGTH` instead of `\get_level VFOA STRENGTH`



99	Transceiver-H-C	Smeter RxA		0	SMTA	A	\get_level STRENGTH
100	Transceiver-H-C	S meter RxB		0	SMTB	B	\get_level VFOB STRENGTH

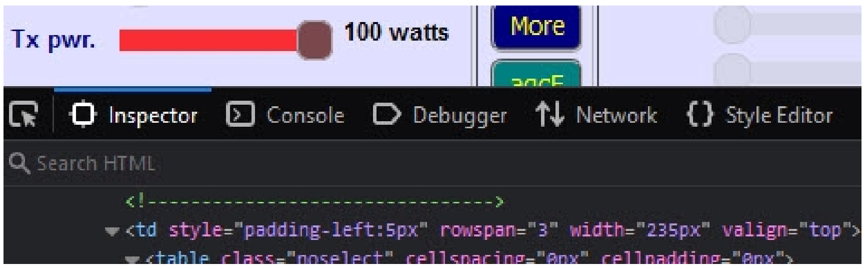
This causes the piWebCAT to repeatedly stop for 2 seconds or so and display the following:



Client-server data timeout. ? disconnection?

The message clears and activity resumes briefly and then the message appears again.
Whats is happeing? ... The RPi server PHP code is sending an illegal meter read to Hamlib rigctld.
The server awaits the meter reading response but it never arrives.
My server PHP code has a two second timeout.
The PHP process sends back an timeout message and then self aborts.
The client code displays the error message on the screen.
We need a method of identifying which of the stream of repetitive commands is causing the problem.

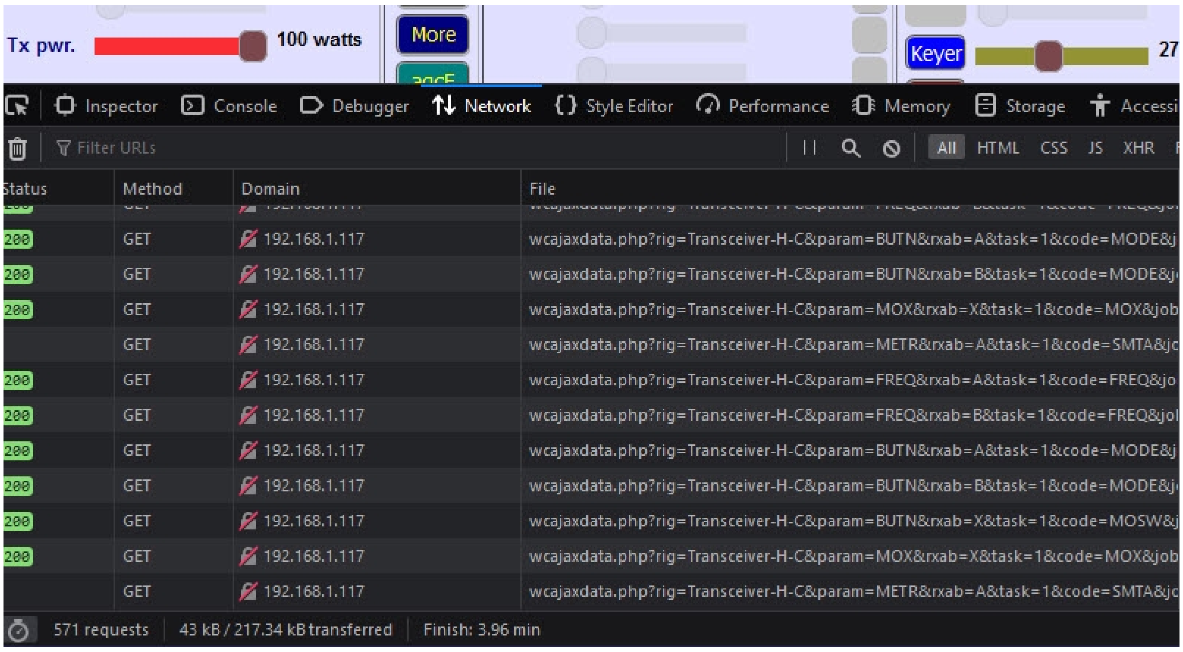
With piWebCAT running, use right mouse **inspect element (Q)**
(... You may have to enlarge the window downwards.)



```
<!-------------------------------->
▾<td style="padding-left:5px" rowspan="3" width="235px" valign="top">
  ▾<table class="noselect" cellspacing="0px" cellpadding="0px">
```

The display will include the above.

Click the **Network** tab



Status	Method	Domain	File
200	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=BUTN&rxab=A&task=1&code=MODE&j
200	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=BUTN&rxab=B&task=1&code=MODE&j
200	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=MOX&rxab=X&task=1&code=MOX&job
	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=METR&rxab=A&task=1&code=SMTA&jc
200	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=FREQ&rxab=A&task=1&code=FREQ&jo
200	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=FREQ&rxab=B&task=1&code=FREQ&jol
200	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=BUTN&rxab=A&task=1&code=MODE&j
200	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=BUTN&rxab=B&task=1&code=MODE&j
200	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=BUTN&rxab=X&task=1&code=MOSW&j
200	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=MOX&rxab=X&task=1&code=MOX&job
	GET	192.168.1.117	wcajaxdata.php?rig=Transceiver-H-C&param=METR&rxab=A&task=1&code=SMTA&jc

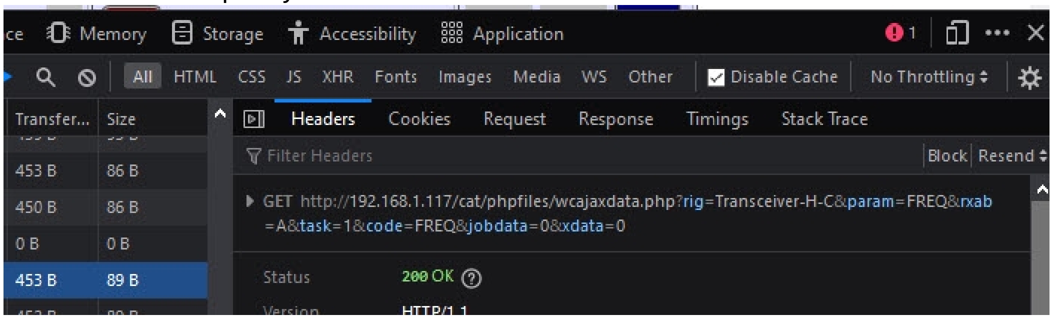
571 requests | 43 kB / 217.34 kB transferred | Finish: 3.96 min

You see a scrolling display of the stream of messages from client to server.
The scrolling keeps briefly stopping ..... on the bottom item of the stream.
You can, if you wish, halt input to the list with the pause button at the top.
-- so long as you capture the item on which the stop occurred.

Examining the bottom item reveals: ...**`param=METR&rxab=A&task=1&code=SMTA`** ....
param=METR is the job type, **rxab=A** means meter for VFOA,
task=1 means read, **code=SMTA** is the client - server link code from the meterhl table.
So - we have identified the problem as occurring in S meter reading
- This should direct you in the first instance to the table where we deliberately created the error.

More....

While we are here .... I have paused the scroll ... we can examine some of the other items in the stream.
If I highlight an item, the window narrows and hides the detail but a second window appear to the right.
I click the headers tab . I can now see the detail of the request again ...
I had selected a frequency read item:



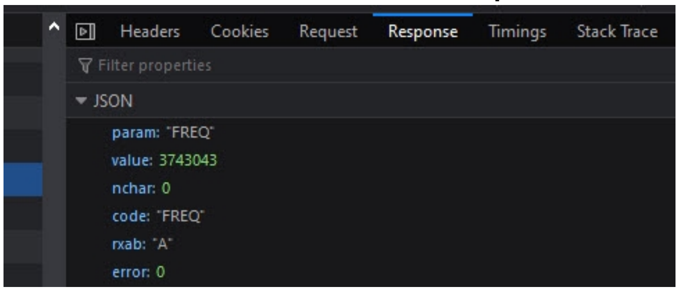
Transfer...	Size
453 B	86 B
450 B	86 B
0 B	0 B
453 B	89 B

GET http://192.168.1.117/cat/phpfiles/wcajaxdata.php?rig=Transceiver-H-C&param=FREQ&rxab=A&task=1&code=FREQ&jobdata=0&xdata=0

Status 200 OK

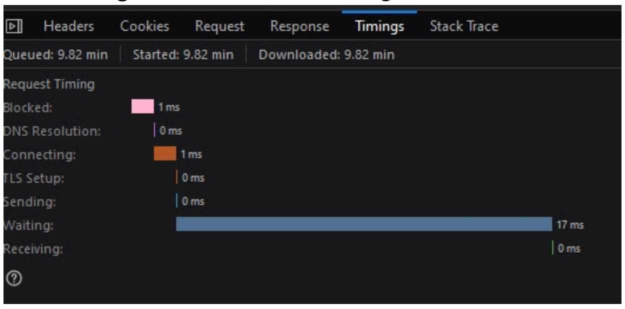
Version HTTP/1.1

You can see param=FREQ raxb=A task=1 code=FREQ
This is a timer driven command to read VFOA frequency - which is performed correctly.
If we switch from the headers tab to the **response** tab, we see the correct response:



```
JSON
  param: "FREQ"
  value: 3743043
  nchar: 0
  code: "FREQ"
  rxab: "A"
  error: 0
```

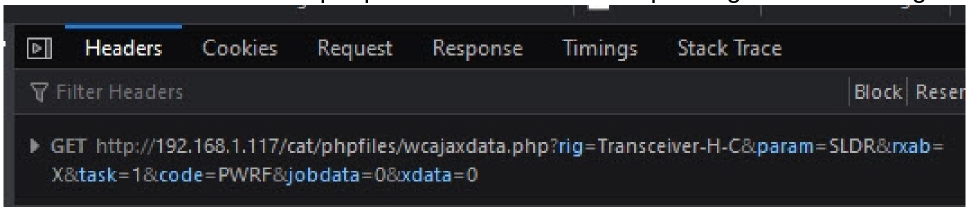
The **timings** tab shows the timing for the command.



```
Queued: 9.82 min | Started: 9.82 min | Downloaded: 9.82 min
Request Timing
Blocked:          1 ms
DNS Resolution:   0 ms
Connecting:       1 ms
TLS Setup:        0 ms
Sending:          0 ms
Waiting:          17 ms
Receiving:        0 ms
```

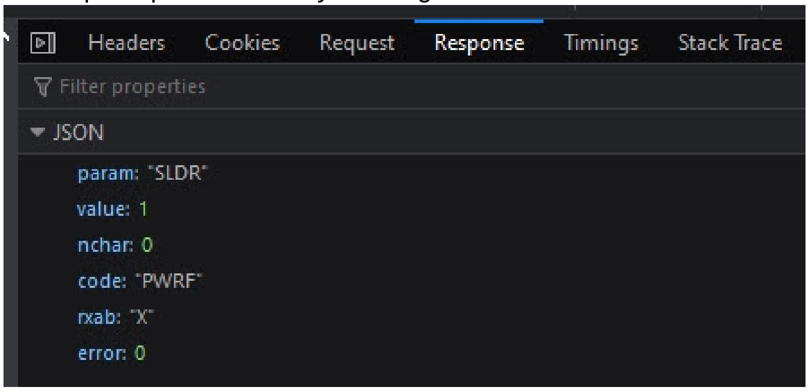
Another example - RF power reading

This is occurring repetitively in the background because I have set the RF power slider with active=S (sync) in order that the slider is kept up to date with the corresponding value on the rig.



```
GET http://192.168.1.117/cat/phpfiles/wcajaxdata.php?rig=Transceiver-H-C&param=SLDR&rxab=
X&task=1&code=PWRF&jobdata=0&xdata=0
```

The response shows a returned value of 1 (This is 100w   ... the data range is 5 to 100w)
I have suggested using **rigctl** at the RPi command line to check data ranges.
This is perhaps another way of doing it!



```
JSON
    param: "SLDR"
    value: 1
    nchar: 0
    code: "PWRF"
    rxab: "X"
    error: 0
```

3.1 piWebCAT database configuration - introduction

The Raspberry Pi's RPiOS operating system has a MariaDb database system.
MariaDB is an open source database system. I shall refer to it as MySQL.

The piWebCAT database's name is **radios.**
It contains 15 tables. At any one time, only 11 are in use because four have separate versions for ASCII and YAESU5 radios and for Icom CI-V (eg: **buttons** or **buttonsciv**, **sliders** or **slidersciv**)

The database is configured for:

- **Internal access** host = localhost
  Used by piWebCAT and its database editors and other RPi database access.
- **External access** via LAN internet host = 192.168.1.17 (changeable), port 3306 (default).
  Used by external database editors or any other program accessing the data via LAN.
  (Note that the default port for MySQL remote access is always 3306)

There are no security issues in controlling a radio (unless someone disagrees!!!)
Therefore internal access and external access share common credentials:
ie: database = **radio** username = **piwebcat** password = **feline**

See Database access

You never have to enter username nor password when running piWebCAT
or its database editor pages.

piWebCAT's built in database editors

There are two web pages accessed by top button bar buttons:

- **Cat config:** Edits nine tables.
  Four tables are different for Icom CIV radios and will change according to radio selection
- **Meter cal.** edits the **metercal** table for S meter and Tx meter calibration..

(The station **log** table is edited only by the log window.)

The editors only present data for the currently selected radio.
They have drop down selectors for some fields, including a list of radios.
The tables have a spreadsheet-like presentation
Editing is directly on to the grid.
The tables can each be exported as a .CSV file which launches in Excel etc.
They can then be printed from Excel.

External database editors

During development, I made extensive use of the PC based **MySQL Front**. This is a free download.
More recently I have tried **HeidiSQL** which is equally effective.
MySQL Front is perhaps faster to use - but then I am very familiar with it.
The application have the ability to export the whole database or individual tables to text SQL files which:

- Are an excellent essential backup of your configuration work.
- Can be imported elsewhere to another piWebCAT installation and so should be able to facilitate sharing of different radio configurations between users.

3.2 piWebCAT - configuration systems.

Development

piWebCAT was first developed on a FTdx101D which used ascii text based CAT commands.
This **ASCII** system is applicable to those radios with text based CAT commands.
Support was then added for Icom CI-V (**CIV**) and for the earlier Yaesu radios which have a 5 byte command system (**YAESU5**).
These systems allow the user to build a configuration data set using the radio's CAT manual.
Not all available radios are supported.

I then discovered **Hamlib,** which had been under gradual development for 10 years.
Hamlib translates a common set of commands into CAT commands for your selected radio.
There are 250 supported radios in the Hamlib database. Each is simply selected by number.
eg: If FTdx101D (#10400) is selected, then **\set_freq Main 3744000** will translate to **FA003744000;**

I initially developed Hamlib support using my IC7000.
Then I added the FTdx101D. My database radio list now has FTdx101D and FTdx101D-H (Hamlib).
The FTdx101D had just been added (mid 2020) in Hamlib version 4.0. I think that it had been done from the CAT manual. It had a lot of bugs.
Much of the development had been done by Michael Black, W9MDB.
There was mutual benefit. I developed a fully working FTdx101d-H configuration and at the same time provided an FTdx101D testbed to help correct all the issues in the Hamlib FTdx101D API.
piWebCAT's existing structure provided an excellent real world CAT development environment.

Hamlib's commands are text -based and so the development was a modification of the already fully developed ASCII system. It was developed after the other systems and so is presented here after the other systems.
References are made to the ASCII system. Some familiarity with the ASCII system is useful.

So we now have four configuration options: **ASCII**, **CIV**, **YAESU5** and **HAMLIB**.

ASCII character based system

piWebCAT was first developed using my Yaesu **FTdx101D** radio.

This uses character based commands which can be freely typed in to the editor fields as text.

eg: **RL012;** = set noise reduction level 0 (VFO A) to 12.

This command system is easy to understand and configure.

CIV - Icom CI-V

I then developed a configuration interface for Icom CI-V CAT control using my Icom **IC7000.**
CI-V is based on hexadecimal and binary coded decimal (BCD) commands.
It required a dedicated configuration interface.
The same system is used in modern Icom radios but with different codes.
I have not applied it to a modern Icom radio - because I do not currently have access to one.
The intention is than the user can use and modify my IC7000 configuration.

YAESU5 - earlier Yaesu radios (FT847, F818, FT1000MP, FT920)

These radios use a **5 byte command format** which is hexadecimal and binary coded decimal.
All outgoing commands are five bytes in length.
Data is returned from the radio in a variety of formats, with lengths from 1 to 28 bytes.

I support these radios using my existing text based configuration system:
The outgoing commands are specified in 10 characters of readable text which is converted in the RPi web server to five data bytes for transmission to the radio.

Returned data is parsed using an **answermask** such as: **#28:21:01:07:xx**
This means: receive **28** bytes, At byte **21** read **01** byte and mask it with hex**07** (= bits 0,1,and 2)

This is how you have to read operating **mode** data for VFO B on a Yaesu FT920 transceiver.
Complex .. yes .. but I have tried to make a readable and easy to interpret user configuration interface.

I initially had this working with my Yaesu **FT847**, which has a very limited command set.
I then purchased a second hand **FT920** which supports many more commands.
(..or should I say that it did so when I had replaced the fried MAX232CWE RS232 interface chip!!.)

The Yaesu5 system is presented as as add-on to the ASCII system .... which you need to look at first.

HAMLIB - control API for radios and rotators

Hamlib translates a common set of commands into CAT commands for your selected radio.
There are 250 supported radios in the Hamlib database. Each is simply selected by number.

The latest Hamlib update is published on https://github.com/Hamlib/Hamlib.
The source code is downloaded to your RPi for initial install or update.
Compilation and installation are straightforward and detailed in this document.
With ongoing development, particularly with new radios, updates can frequently be on a daily basis.
Hamlib is installed on the downloaded RPi SD card image.

piWebCAT uses **rigctl** and **rigctld**.

rigctl is a command line facility that can be used to issue single CAT commands from the RPi terminal.
To open the connection to the rig, the typical syntax is:

```
$  rigctl -m 1040  -s 38400 -r /dev/ttyAMA0 --vfo
```

1040 is the Hamlib id of FTdx101D. 38400 is the baudrate. --vfo is to set dual VO mode
/dev/ttyAMA0 is piWebCAT's configured serial port (or use **/dev/ttyUSB0** for a USB conection)
The response is: **Rig command:**

You can then enter **rigctl** commands, eg **\set_level Main NR 0.4**
These will be translated to FTdx101D commands and sent to the rig.
This is a very useful way of testing Hamlib commands before configuring them into piWebCAT.

rigctld is used by piWebCAT

It is started by piWebCAT at startup and provides a **serial TCP socket** on **localhost:4532**.
piWebCAT's PHP webserver code then sends rigctld commands to this socket.

Unsupported commands: Not all of a radio's hundreds of less frequently used CAT commands are supported.
To deal with this, **rigctl** has a **\send_cmd_rx** command whereby you can send the radio's actual CAT command.

eg: For FTdx101D For full/semi-breakin switching, I use **\send_cmd_rx EX020111#; 0**
(# = 1 for full, 0 for semi)

3.3 A configuration strategy for your radio.

If you have a modern Yaesu rig, you may choose to begin by modifying my
FTdx101D setup. If so, you can intially leave all the records as **rig** = FTdx101D.

If you have a modern Icom rig, I think the configuration will be very similar but the
command details will have to be changed.

Alternatively, you can use the HAMLIB option.
Hamlib is a programming interface the converts a generic command set in the command set for
a selected one of 250 radios. The selection is by a reference number.
The learning guide in this manual progressively builds up a Hamlib configuration.

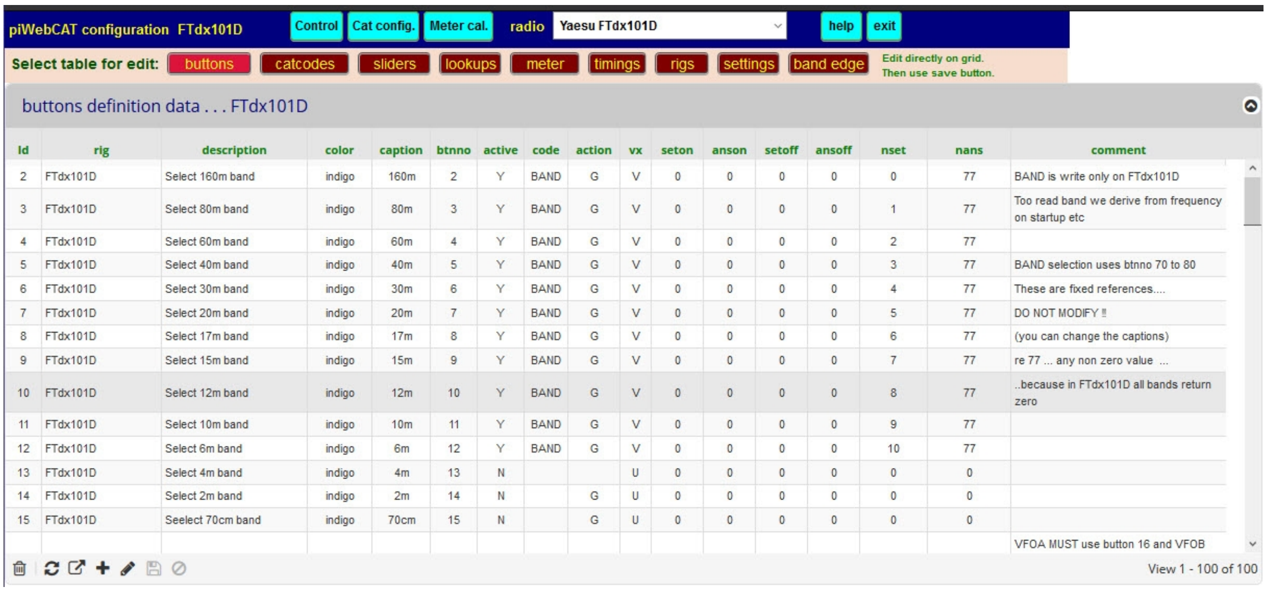
If you want to build a radio configuration from scratch, then I suggest that you leave my radios
in place as a guide and template.
You can then gradually build in your radio as a new addition.

- You can have dual access to the database:
  ie: view my configuration as a template example with MySQL Front on a PC
  and simultaneously use piWebCAT's built in editor for adding the new radio
  (or vica versa).

- You can build in one command or command group at a time and test it.

3.4 piWebCAT - database editor - operation

The opening editor window for a Yaesu FTdx101D radio is shown below.
The grids are generated using **phpGrid.**



buttons definition data . . . FTdx101D

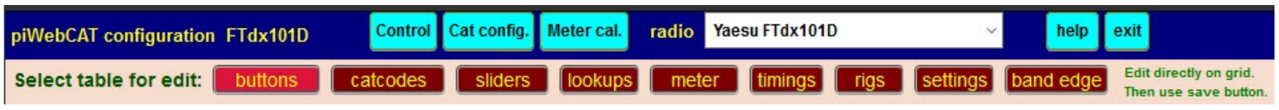
Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset	nans	comment
2	FTdx101D	Select 160m band	indigo	160m	2	Y	BAND	G	V	0	0	0	0	0	77	BAND is write only on FTdx101D
3	FTdx101D	Select 80m band	indigo	80m	3	Y	BAND	G	V	0	0	0	0	1	77	Too read band we derive from frequency on startup etc
4	FTdx101D	Select 60m band	indigo	60m	4	Y	BAND	G	V	0	0	0	0	2	77	
5	FTdx101D	Select 40m band	indigo	40m	5	Y	BAND	G	V	0	0	0	0	3	77	BAND selection uses btnno 70 to 80
6	FTdx101D	Select 30m band	indigo	30m	6	Y	BAND	G	V	0	0	0	0	4	77	These are fixed references....
7	FTdx101D	Select 20m band	indigo	20m	7	Y	BAND	G	V	0	0	0	0	5	77	DO NOT MODIFY ‼
8	FTdx101D	Select 17m band	indigo	17m	8	Y	BAND	G	V	0	0	0	0	6	77	(you can change the captions)
9	FTdx101D	Select 15m band	indigo	15m	9	Y	BAND	G	V	0	0	0	0	7	77	re 77 ... any non zero value ...
10	FTdx101D	Select 12m band	indigo	12m	10	Y	BAND	G	V	0	0	0	0	8	77	..because in FTdx101D all bands return zero
11	FTdx101D	Select 10m band	indigo	10m	11	Y	BAND	G	V	0	0	0	0	9	77	
12	FTdx101D	Select 6m band	indigo	6m	12	Y	BAND	G	V	0	0	0	0	10	77	
13	FTdx101D	Select 4m band	indigo	4m	13	N			U	0	0	0	0	0	0	
14	FTdx101D	Select 2m band	indigo	2m	14	N		G	U	0	0	0	0	0	0	
15	FTdx101D	Seelect 70cm band	indigo	70cm	15	N		G	U	0	0	0	0	0	0	
																VFOA MUST use button 16 and VFOB

View 1 - 100 of 100

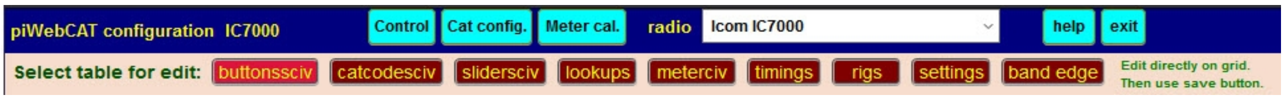
The dark blue top bar is common to all three web pages.
The radio selector shows FTdx101D. This selection is stored in the database **settings** table and so is retained on closure.

Each table has a selection button. Four tables are different between ASCII radios and Icom CI-V radios. The table selection changes according radio selector change.

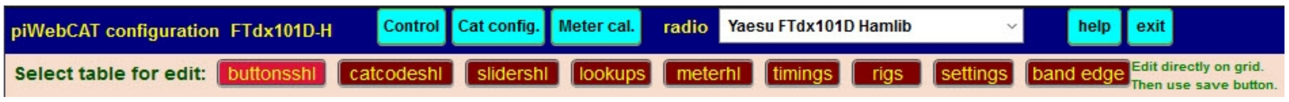
This is best illustrated by the button bars:



ASCII radios - Yaesu, Kenwood, Elecraft and older Yaesu 5 byte command radios



CI-V radios Icom



Hamlib radios

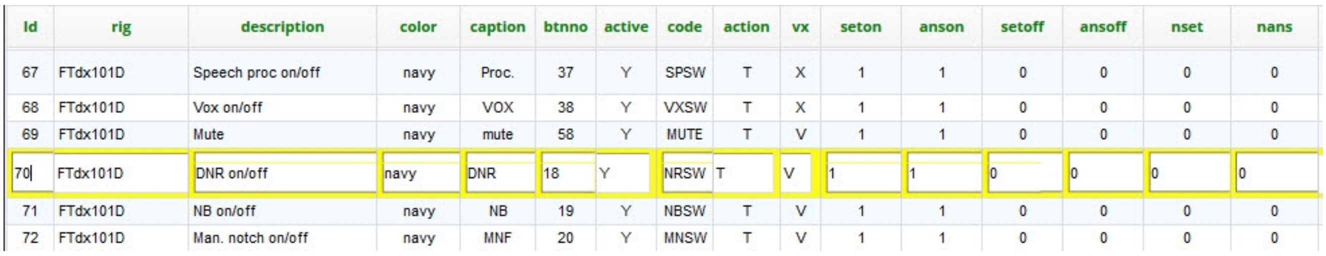
phpGrid control buttons

At the bottom of the grid are the phpGrid control buttons for editing and for export to CSV file.

🗑	Deletes highlighted record.
🔄	Refresh grid from database.
↗	Export current table to a .CSV file - launches in Excel -> for optional printing.
➕	Add a new record.
✏	Edit highlighted record - OR just click on a cell.
💾	Save the edited record.
⊘	Abort editing.

In line editing

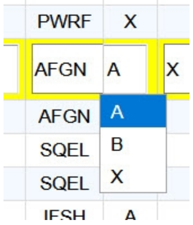
Editing is done directly on the grid - Clicking on a record opens the whole record for editing thus:

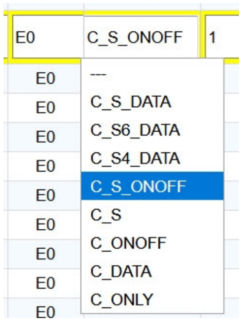


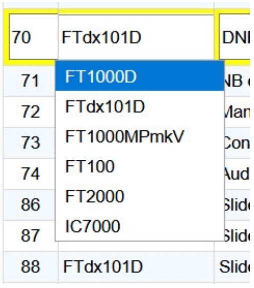
Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset	nans
67	FTdx101D	Speech proc on/off	navy	Proc.	37	Y	SPSW	T	X	1	1	0	0	0	0
68	FTdx101D	Vox on/off	navy	VOX	38	Y	VXSW	T	X	1	1	0	0	0	0
69	FTdx101D	Mute	navy	mute	58	Y	MUTE	T	V	1	1	0	0	0	0
70	FTdx101D	DNR on/off	navy	DNR	18	Y	NRSW	T	V	1	1	0	0	0	0
71	FTdx101D	NB on/off	navy	NB	19	Y	NBSW	T	V	1	1	0	0	0	0
72	FTdx101D	Man. notch on/off	navy	MNF	20	Y	MNSW	T	V	1	1	0	0	0	0

After editing, the changes must be saved back to the database with save button.

Where possible, field editors have been configured as drop down lists:







The **abx** and **cmdtype** lists shown to the left and have fixed content.
The **rigs** list is derived from the **rigs** table in the database.
The radio's name is stored as a character string and appears in every table as a link for the radio. Spellings must be identical.
This is ensured by typing the radios name only once into the **rigs** table from where it is then selected unchanged into the other tables.

The field **code** is the link between client and server in making data requests.
The **code** field is not offered as a drop down list because the user my need to create new **code** items to control functions that I have not so far supported.

3.5 piWebCAT - Database tables - overview

For ASCII radios (Yaesu, Kenwood, Elekraft) and **YAESU5** radios(FT920, FT747 etc) we have:

• **settings**	A single record with fields for the current radio (**rig**) and for fast, medium and slow tuning rates for mouse drag and thumbwheel tuning. **(mwF**, **mwM** etc)
• **rigs**	A table of radios with **connection** (via Encoder CAT or direct) **catcomms** (CIV or ASCII) and serial interface settings.
• **timing**	The repeat intervals in ms for piWebCAT's repetitive tasks. eg: read meter, send frequency etc
• **metercal**	The calibration table for six meters per radio (S meter and 5 transmit meters) Each calibration is 20 points and is interpolated when in use. **metercal** has its own editor page.
• **buttons**	piWebcat has 66 buttons. This table contains the settings that need to reside on the client. (They are extracted from the database at startup)
• **catcodes**	Settings for buttons that need to reside on the server, for communication with the radio. Also for frequency read and write with the radio.
• **sliders**	All the slider data - ie some extracted to the client at startup and some used on he server for communication with the radio.
• **meter**	Meter data for S meter and Tx meters. Some is transferred to the client at startup, some is used on the server for communication with the radio.
• **lookups**	Used for the numeric display of some slider settings when there is a non-linear relationship between the CAT data value and displayed value.
• **bands**	Specify band edges for band display selection (NB: tuner scales are fixed)
• **log**	Data of piWebCAT's amateur radio station log.

For Icom CI-V radios
Four tables are different: **buttonsciv**, **slidersciv, catcodesciv** and **meterciv**.
The above descriptions for ASCII radios still apply.

For radios configured using **HAMLIB**.
Four tables are different: **buttonshl**, **slidershl, catcodeshl** and **meterhl**.
The above descriptions for ASCII radios still apply.

rig
The **rig** field is a text identifier for the radio, eg: 'FTdx101D', 'IC7000'.
It is repeated in every table record for a radio.
The piWebCAT editors only display the currently selected radio.
If the **rig** field is mis-spelt or corrupted then the record will not be visible.
Fortunately, you type it once in the **rigs** table and thereafter select it from a list.
In my example database I have CIV and HAMLIB versions for the IC7000.
They appear as two different radios, IC7000 and IC7000-H. (You can choose your own labels)
Likewise, I have FTdx101D and FTdx101D-H.

To set up for a new radio:
*You must first define your radio by adding it as a record in the **rigs** table.*
*Then select the rig as the current rig in the **settings** table by editing the settings table OR by using the top bar selector.*
(The settings table has only one record which holds the current rig and other settings.)
*Now, when you create a new record (button, slider etc) , your new radio will automatically appear in the **rig** field.*

ASCII / CI-V / YAESU5 / HAMLIB
The following pages discuss each table.
There is repetition because of the common fields between the different configuration systems.
eg: **buttons** / **buttonsciv** / **buttonshl** etc

3.6 Button and slider numbering

Please see also Button operation reserved buttons reserved codes

Button and slider controls are allocated a fixed numeric identifier.
The numbers appear in the button and slider database tables.

- **btnno** is used to identify the button's onClick() events.
  It also appears in a fixed coded table to link **bttno** to the HTML **id** of the button.
- **sliderno** is used to identify the slider's onChange() event.
  It also appears in a fixed coded table to link **sliderno** to the HTML id of the slider and of the adjacent caption and numeric value text.

You cannot change the number of an on-screen control.
*You can change a **btnno** or **sliderno** value in an item your configuration table in order to move the configured function to a control in a different position in the screen layout.*

Below is an extract from the internal slider table in the sliders.js program module.
You don't have to interact with these labels. They are shown here to illustrate the underlying structure and the fact that a few controls are of fixed position

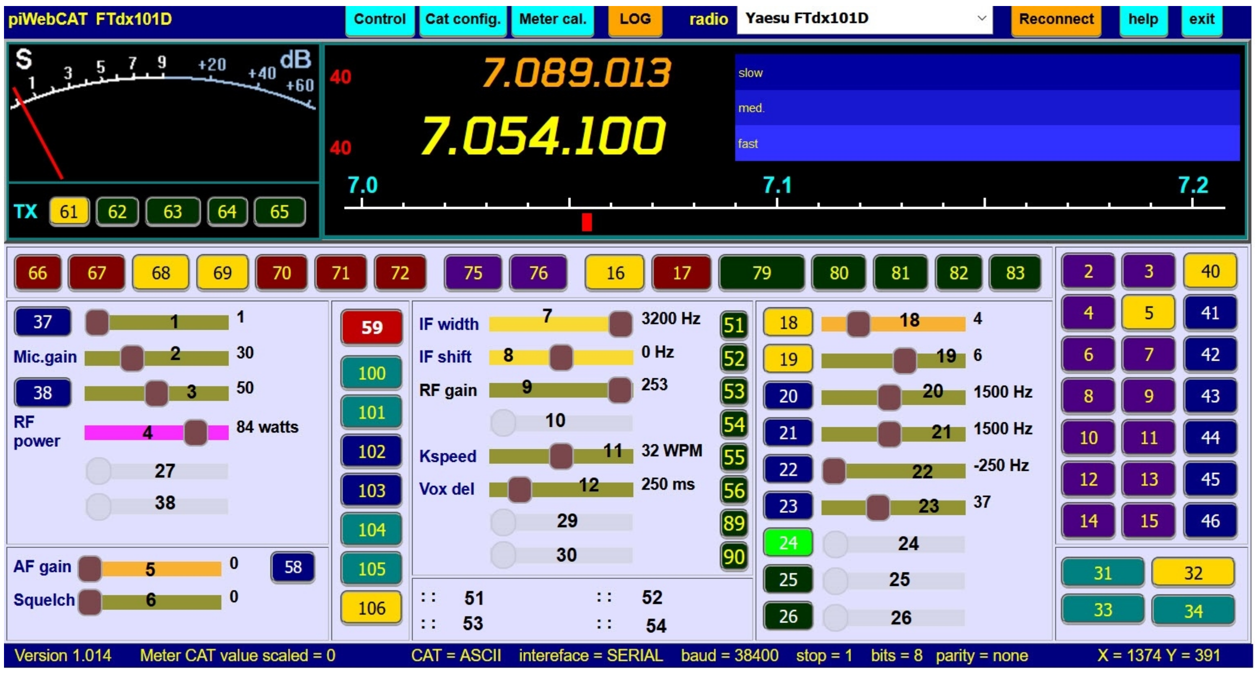
```
[SLIDER_IF_WIDTH,  "sliderIfWidth",  "textIfWidth",  "capIfWidth"],
[SLIDER_008,       "slider0008",     "text008",      "cap008"],
[SLIDER_009,       "slider009",      "text009",      "cap009"]
```

rest have numeric internal labels, eg: `SLIDER_008`.

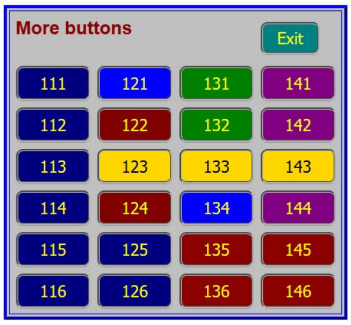
`SLIDER_008`	is **sliderno**
`slider008`	is the id of the slider
`text008`	is the id of the text numeric value to the right of the slider.
`cap008`	is the id of the caption to the left of the slider.

The functions, captions, button colours etc are changed at startup by the database configuration.

The image below is an essential configuration tool. It shows the controls' **sliderno** or **btnno** values.
To create it, I used a working layout and changed the captions to the controls' fixed numeric id numbers.
The text labels are simply residual from the layout that I modified.



Version 1.014   Meter CAT value scaled = 0   CAT = ASCII   intereface = SERIAL   baud = 38400   stop = 1   bits = 8   parity = none   X = 1374 Y = 391



'Sliders' **51** to **54** are text display items. They are handled in the **sliders**, **slidersciv** and **slidershl** tables. See Text display box.

The popup window with 24 extra buttons is launched with a button whose **code** field is set to **MORE**.
The extra buttons have the same status and configuration process as the main buttons.

3.7 piWebCAT - setup - Important reading

piWebCAT's control actions are NOT read-modify-write (unlike my EncoderCAT project)
The controls are synchronised to the radio's settings at startup, on band change
and by the reload button.
This uses the READ and ANSWER configurations to request the data and interpret the answer.
Subsequent button or slider action commands use SET configurations with the new position of a slider,
or from known buttons states. ie: only the SET commands are used.... there is no feedback.
(eg: for toggled buttons, piWebCAT uses the button's remembered on or off state
to determine the new state.)

The database is named **radios**.

Database tables

It contains 10 tables:

- **buttons**, **sliders, catcodes** and **meter** are used for **ASCII** and **YAESU5** radios.
- **buttonsciv**, **slidersciv**, **catcodesciv** and **meterciv** are for **Icom CI-V** radios.
- **buttonshl**, **slidersshl**, **catcodeshl** and **meterhl** are used for **HAMLIB** configurations.
- **rigs**, **settings**, **timings**, **lookups, metercal and bands** are used by all radios.
- **log** stores the station log.

The tables can hold records for multiple radios.
A radio is identified by the **rig** field. This is a character string that must spelled correctly, otherwise the record will disappear from piWebCAT's editor.

What not to change!

piWebCAT's configuration system is flexible in that it allows the user to create new control items and to name their control codes. However, their are certain names / fields which must not be changed.
These are listed below:

- The 91 buttons and 27 sliders have a fixed allocated numeric references: **btnno** and **sliderno**
  These are identified on a control page layout on the preceding page.
  They are hard coded in the software.
- **code** fields are the link between client data and server data, eg: NRSW, VXSW etc
  Many of these could be changed provided that the same name is used in client and server (see below)
  However, certain code items are used in the software and so must not be changed:
  ie: BAND, MODE, VFO, FREQ, SWAP, SPLT, ATOB, BTOA, MOX, TXME, MORE, MPAD,
  CWME, MECH, FRUP, FRDN, RPGO, CWRP, PWRF.
- The available values of **active, abx, vx, lookup, action, numchar, usecal, meter, catcomms, connection, vfobvis, afswap, disable, cmdtype** are fixed. (eg: some are just Y / N)
  These are in fact all offered by drop down selectors in piWebCAT's editors ... so you can't go wrong
  - but they have to be typed when you use an external editor such as MySQL Front.

Tables: buttons, catcodes, sliders, meter

The above are ASCII radios tables.
The following equally applies to CIV tables: bttonsciv, catcodesciv, slidersicv, meterciv)

I give FTdx101D examples.

Much of this discussion relates to the fact that one slider on the client will control two radio parameters, once in each receiver according to current VFO A/B selection.

(NB: The FTdx101D has two completely separate receiver modules and so most receiver control setting are set separately for each receiver. This is not so in all 'dual VFO' radios.)

sliders

*The **sliders** table holds both client and server data.*
*(Whereas for buttons, client and server data are in separate tables: buttons and **catcodes**).*

The **sliders** table contains either one or two records for every slider.

Two sliders table records: for A and B VFOs / receivers: **abx** = A or **abx** = B
Receiver A: .
eg: Noise reduction level 0 -15.
One record has: **abx** = A **vx = V** and CAT command masks for VFO A ie: RL0; RL0tu; RL0tu;
One record has: **abx** = B **vx = V** and CAT command masks for VFO B ie:RL1; RL1tu; RL1tu;
All other fields are identical between the two records. (except your non-functional description text)

The command **masks** and **abx** are used on the server
Most of the remaining fields are extracted to the client at startup (text value formatting etc.)
Both client and server will have the linking code: **NRLV**

Note the vx field
This can have value: V, X or U. It is copied to the client at startup.
It is set to V if the associated receiver setting has different values for VFO A and VFO B.
By this I mean that the radio holds a different value for each VFO.
*Setting **vx** = **V** instructs the client to message the server with the identity of the current VFO (A or B).*
*Setting **vx = V** instructs the client to store the latest value for each VFO* See: Dual VFO switching

At startup, a message is sent to the server to generate a data array of client-required fields for all the sliders.
Only one record is needed for the pair of sliders . so the first encountered is extracted and the second is ignored
(The data for the client is identical between the records anyway)

If one sliders table record: for sliders that are not VFO A/B dependent
eg: Vox gain, RF power
For these, **abx** = X **vx = X**

If **abx** = A or B then the client sends A or B according to current VFO selection.
If **abx** = X then the client sends abx = X

buttons

For buttons, we have two separate tables:
- **catcodes** for the server data ***NB catcodes also records for FREQ (frequency read and write)***

Why different to sliders? ie: two tables rather than one? - This is partly developmental.
- but also because we can have large groups of buttons on the client linking to either one or two server records to transmit the code of the pressed button in the group.

The **buttons** table has an **action** field with values: U, S, T G, M, R (drop down list selector).
These act as follows

- **U** Unused
- **S** Single action (momentary action) button (no data), eg: swap VFOs
- **T** Toggled buttons. Changes state at every press, On/Off - illuminates when on.
  Client stores current state in order to send appropriately on or off when pressed.
  Uses fields **von** and **voff** (usually 1 and 0)
- **G** Grouped. A number of buttons in a group. Only one selected at a time.
  All have **action** = G and a common **code** field
  eg: Attenuator: 0dB, 6db, 12dB, 18dB .. send (and receive) data = 0 to 3 in nans and nset
  Common code field is **ATTN**. **ATTN** is the link to a pair of **catcodes** records with **code** = **ATTN**,
  one with **abx** = A for VFO A and one with **abx** = B for VFO B.
- **M** The five grouped Tx meter records - these specify which Tx metering value will be read from
  from the radios by the repetitive meter reading process on transmit.
  The captions of these five buttons are in the **meter** table. They are user configurable to allow
  the user to select which of the available metering options to offer on the buttons.
- **R** The reset buttons for nine sliders. They are internally linked to the adjacent slider.
  They set the slider to the **def** value extracted to the client from the **sliders** table.

To repeat .... for Grouped buttons:
*We have a group of buttons in the **buttons** table, each with a shared value for the **code** field.*
*The group has **action** = G and a common **code** (eg: **ATTN**)*
*We have two records in **catcodes** with **code** = ATTN and **abx** = a for VFOa and **abx** = B for VFO B.*

*Note that the client sends **code** = ATTN **and** abx **= the current VFO ie: A or B.***
*(Setting **vx** = **V** makes instructs the client to do this, ie: send current VFO identity)*

Non A/B buttons

eg: Vox on/off, processor on/off - VFOA, VFOB, SWAP A/B, A >B, B>A, Split
These buttons have a single record in the catcodes table with **abx = X**

The vx field in the buttons table

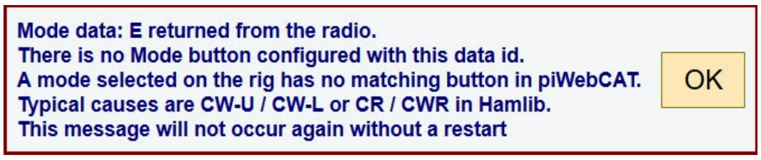
This has values U, V or X.
It controls how data is sent to client (ie: whether to send current VFO selection (A / B)
and if and how to store the buttons state on the client for fast VFO switching.

- **U** Prevents participation in control state storage and retrieval on client.
  May or may not send data to server.
  Used for: Band buttons, meter buttons and slider reset buttons.
- **V** WIll send **abx** = A or B to server, according to current VFO selection.
  Enables buttons to participate in state storage and then retrieval on VFO change
  with stored A and B values.
- **X** Will send **abx = X** to the server. Not A / B dependant. Not stored for VFO A/B switching.

MODE button and button groups caution.

This discusses a button group where we have not provided a button for all the options on the rig.

Example: My FTdx101D has fifteen modes.
I provided only seven mode buttons configured as a group linked to seven of the fifteen modes.
If I do all my mode selection on piWebCAT, then there is no problem.
If, on the rig, I select a mode that has no corresponding piWebCAT button, then the message
box below will appear as piWebCAT starts or when the such a mode is first selected on the rig.
The message only appears once and then is suppressed until you restart piWebCAT.


Mode data: E returned from the radio.
There is no Mode button configured with this data id.
A mode selected on the rig has no matching button in piWebCAT.
Typical causes are CW-U / CW-L or CR / CWR in Hamlib.
This message will not occur again without a restart

A similar generic message box will appear if the problem arises in other button groups.
The box occurs on start up OR the box appears if the unrecognised condition is subsequently selected
on the rig, but only if the button group has **active=S** which is sync mode for regular updates.

FTdx101D AGC speed has a particular problem:
I provide three buttons, fast, medium and slow code 1,2 and 3.
The received AGC data from the rig is 1,2 and 3 AND 4 = fast auto, 5 = med.auto and 6 = slow auto.

piWebCAT has a solution to this. The matching code for grouped button received data is in field **nans**.
***nans** can in fact hold multiple values, all of which result in selection of the associated button.*
Multiple values are separated by an **|** character, eg: **4 | 5 | 6 which means = 4 or 5 or 6.**
Spaces are optional, ie: **4|5|6** is also valid. (**|** is the OR symbol in C, PHP and javascript languages)

So we have two options:

1. Use the three buttons, set **nans** equal to: **1 | 4** for fast, **2 | 5** for medium and **3 | 6** for slow.
2. Use four buttons: auto, fast, medium and slow and set nans equal to **4 | 5 | 6** for the auto button.

For the piWebCAT direct configuration I use option 2.
With Hamlib and FTdx101D, **rigctld** returns **6** for all auto setttings (so I set **nans = 6** for the auto button.)

To prevent the auto button sending code 0 (= AGC OFF!), I set **nset = 'xxx'** which blocks any action.

3.8 Reserved button numbers .... buttons that have a hard coded function.

References are to **buttons** and **catcodes** tables.
All of the following discussion is also applicable to **buttonsciv** and **catcodesciv** for Icom.

Every button (including the 24 buttons on the popup window) has a unique numeric identifier.
See Button and Slider numbering

In code module, buttons.js the button number are assigned text labels, eg:

```
        ............
        const BTN_6M          = 12;
        const BTN_4M          = 13;
        const BTN_2M          = 14;
        const BTN_70CM = 15;
        const BTN_VFOA = 16;
        const BTN_VFOB = 17;
        const BTN_DNR  = 18;
        const BTN_NB          = 19;
        const BTN_NOTCH       = 20;
        ............
```

These labels refer to button usage in my initial FTdx101D configuration.
They are purely internal references in the software for design convenience.
You can configure the system without being aware of them.

Most buttons (and sliders) can be configured to a function, caption and colour of your choice.
However, some buttons have hard-coded dedicated functions. These are detailed below.

The **buttons** table data is loaded to the client at startup.
The **catcodes** table data is read from the data 'on the fly' by PHP server code.
The primary link between the **buttons** record and the **catcodes** record is the **code** field.
Most code field items can be your choice. However some are fixed for use by the software.

The other important links is the buttons **vx** and the catcodes **abx** field ...
See next section: vx and abx

(Note that a **code** field is also in the **sliders** table.
With **sliders**, both client and server are in a singe table, but the use of **code, vx, abx** is the same.)

Fixed button numbers:

VFO control buttons:

16	VFO A	Use code = VFO
17	VFO B	Use coded = VFO
79	Swap A/B	No direct server action ... software link to VFO A and B buttons.

The intended configuration with the two receivers or two independent VFOs is as
my FTdx101d example:

Firstly, note that VFO selection is not a function of the separate VFOs
It *selects* the VFOs . It uses **vx = X** and **abx = X.**

We configure buttons 16 and 17 as a group of two (**action = G**) and use **code = VFO.**
The buttons each send the data for their specific VFO selection to a single **catcodes** record
on the server. This generates the VFO switching commands using the button data.

Button 79 action is interception in software and toggles between buttons 16 and 17 action.
Thus we have two ways of doing the same switching job, ie click the VFO A or B buttons or
toggle them with the Swap button.

For this to work, the radio's CAT controls need to have read and write VFO selector commands.
The FT920 doesn't appear to have this (contrary to what the manual says.)
I therefore just have a SWAP button .... and it cannot be button 79 with its built in toggling action.
I simply choose another button as Swap and disable buttons 16, 17 and 97.

Tx meter buttons.

These are:

61	Tx meter A	eg: power	code = PWRM
62	Tx meter B	eg: ALC	code = ALCM
63	Tx meter C	eg: Compression	code = CMPM
64	Tx meter D	eg: IDD	code = IDDM
65	Tx meter E	eg: SWR	code = SWRM

These buttons do not communicate with the radio,
They select which metering command shall be sent to the radios on transmit.
They all must have code = TXME.
Their captions are specified in the meter table.
The button number (61, 62 etc) is the link to the meter table (not the code)

Band buttons (not fixed in the code)

These in groups on the right of the window.
A group is defined by all its buttons having action = G and the same value for the code field.
The code field must be BAND (see below)
The buttons numbers are not actually fixed ... but it is suggested that you use my original allocation.
(but you can rearrange them if you wish)

2 = 160m	3 = 80m	4 = 60m	5 = 40m	6 = 30m	7 = 20m	8 = 17m
9 = 15m	10 - 12m	11 = 10m	12 = 6m	13 = 4m	14 = 2m	15 = 70cm

Mode buttons (not fixed in the code)

These in groups on the right of the window.
A group is defined by all its buttons having action = G and the same value for the code field.
The code field must be MODE (see below)
The button numbers are NOT fixed.
On a 160m - 6m rig, band buttons 13, 14 and 15 are not used and so could used for extra modes.
(Change the colour to navy to match the other other mode buttons)

3.9 Reserved *code* field values

References here are to the **buttons** and **catcodes** tables.
All of the following discussion is also applicable to tables **buttonsciv** and **catcodesciv** for Icom CI-V and to tables **buttonshl** and **slidershl** for Hamlib configuration.

The **buttons** table data is loaded to the client at startup.
The **catcodes** table data is read from the data 'on the fly' by PHP server code.
The primary link between the **buttons** record and the **catcodes** record is the **code** field.
Most **code** field items can be your choice. However some are fixed for use by the software.

The other important links is the buttons **vx** and the catcodes **abx** field ...
See next section: vx and abx

(Note than a **code** field is also in the **sliders** table.
WIth **sliders**, both client and server date are in a singe table, but the use of **code, vx, abx** is the same.)

List of reserved code field values:

• **TXME**	Used on Tx meter buttons - see above These buttons do not communicate with server. They specify which meter to present on transmit.
• **VFO**	Used on VFO A and VFO B buttons (grouped) - communicates with single **catcodes** record.
• **SPLT**	Use on split buttons - Allows piWebCAT to detect split frequency working
• **BAND**	Use on all the band selector buttons ... single **catcodes** record .. only acts on current VFO.
• **MODE**	Use on all mode selector buttons ... and in **catcode**s records (may have separate VFO A and B records)
• **MORE**	Launches the 24 extra button popup.
• **MPAD**	Launches the memory keypad popup.
• **MECH**	Use this for memory channel selection. Not actually essential for individual memory buttons ... but the memory keypad uses **MECH** and so must be supported by a **catcodes** memory selector with **code = MECH**. (.... No point in having a separate **catcodes** record for keypad and individual buttons!!)
• **MTOV**	On my FTdx101D and FT920, **MECH** commands select the memory *AND load it to the VFO*. On my IC7000, a separate *'load to VFO'* command needs to follow memory channel selection. piWebCAT will look for a catcodesciv **MTOV** command after **MECH**. If found, will automatically issue your configured **MTOV** command. **Works well on my IC7000. I don't know if it's needed on more modern Icom rigs.**
• **MOX**	Must be used for MOX button.
• **FREQ**	Must be used for VFO A/B frequency read and write. Generated by piWebCAT tuner - so no **buttons** nor **sliders** record. Just server **catcodes** records However, for the FT920, I could find no band switching command - so the band buttons use **code = FREQ** and send a start frequency for each band. (piWebCAT then follows the radio's band change and displays the new tuning scale) .
• **FRUP**	Sends a **code = FREQ** command to the server to step the frequency up.
• **FRDN**	Send a **code = FREQ** to the server to step the frequency down.
• **DBUG**	A buttons with **code = DBUG** launches a debug popup window (draggable). It can be configured on any spare button using **active = Y** and **action = S**. The window displays nine debug items. An item is set using: **`pwcDebug(itemno, label, data);`** See debug window

3.10 Please read: The vx and abx fields restated!!

The use of the **vx** and **abx** data fields is discussed elsewhere.
It is easy to initially overlook their correct use ...
(in view of the large amount of other configuration information!)

I guess that I feel the need to restate **vx** and **abx** use because it took me quite a number of iterations to get this correct.
Part of the problem was not fully understanding the behaviour of the IC-7000 in relation to which receiver settings are VFO A or B specific.

piWebCAT loads up to 90 button states and 22 sliders states at start up.
Settings may be different for different bands and/or between VFO A and VFO B.
This will vary between radios.
piWebCAT can store the latest setting for each of these controls. This avoids having to reload settings from the radio on VFO A/B switching.
In order to do this, it needs to know which settings can change with VFO change.
(If the settings are VFO specific, then it stores separate A and B VFO settings)

IC-7000 IC-7610

My IC-7000 piWebCAT configuration controls the receiver parameters including those listed below.
The details given in the table refer to the ***behaviour of the radio***,

DNR on/off and DNR level	single settings	toggling button and slider	abx = X	vx=X
NB on/off and level	single settings	toggling button and slider	abx = X	vx=X
Manual notch on/off,tune	single settings	toggling button and slider	abx = X	vx=X
Auto notch on/off	single setting	toggling button	abx = X	vx=X
AGC slow/med/fast	single setting	3 button group	abx = X	vx=X
IF width	separate for each VFO	slider	abx =A & B	vx=V
Pre amp on.off	separate for each VFO	2 button group (IPO/amp)	abx =A & B	vx=V
Attenuator on/off	separate for each VFO	2 button group (Att0 / Att12	abx =A & B	vx=V

Buttons: If separate per VFO then needs two catcodesciv records: one for A and one for B
(These two records are identical ... but not so for other radios (Yaesu) ... compatibility need)
If there is group of buttons, then one buttonsciv record per button.
(buttonsciv is client data, catcodesciv is server data)

Sliders: Two records in slidersciv if 'separate'.
(slidersciv has both client and server data)

The **IC-7000** manual provides CAT codes for all the above but does not appear to give any indication of whether single or separate per VFO. - I had to experiment.

The CI-V manual for the current, dual receiver **IC-7610** does not appear to state which settings are separate per VFO. piWebCAT needs to know this information. You have to experiment !

Yaesu FTdx101D -

Most receiver settings are separate between the two VFOs.
So most settings have two (A & B) catcodes and sliders table entries
and have buttons.**vx** = **V** **catcodes.abx = A** and **B** **sliders.vx = V** and **sliders.abx** = **A** and **B**.

Why this level of complexity??
Reason - Buttons and slider related settngs change on VFO switching and would need reloading. This perhaps takes longer on a web server based solution compared to direct connection from PC to radio.
I therefore store the latest settings against each VFO to avoid a reload. I need to know which settings change.

3.11 piWebCAT - Memory selection.

The FTdx101D has a CAT command: **MT** to program a memory channel.
It sends channel number, VFO_A frequency, clarifier settings, mode, CTCSS data, simplex/shift.
All this is sent in one command, 50 characters in length. piWebCAT cannot handle this in any useful way.
(It could send a fixed long command but cannot edit so many components within the command.)

piWebCAT can select and apply memories that have been *pre-programmed in the radio*.
There are two ways to do this:

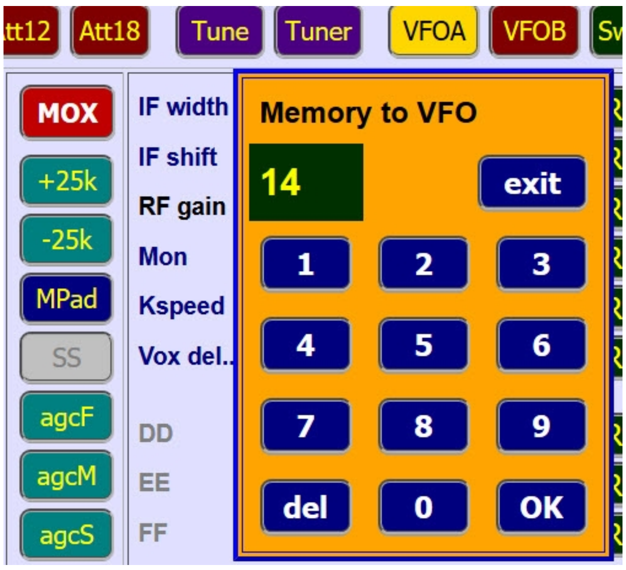
- **Memory channel buttons** each having a fixed, single channel assignment. **cmdtype = MECH** (fixed)
- A button to launch a numeric, **memory selecting keypad**. **cmdtype = MPAD** (fixed)

Both of the above commands are followed one second later by a 'pseudo button' on the client sending an automatic *'copy selected memory to current VFO'* command (**cmdtype = MTOV**).
If MTOV is not needed (eg: Yaesu FTdx101D), then it if it is not configured in the **catcodes** table it will simply be lost.

*For the IC-7000 (and ? other Icom radios), the automatic **MTOV** command is needed because the 'memory select' command selects the memory but does not apply it to the VFO.*

Single channel buttons eg: for calling channels.
These are programmed like any other single press buttons (**action = S**).
Each such button has a record in the **buttons** (or **buttonsciv**) table.
You can configure multiple buttons sharing a common **catcodes** (or **catcodesciv** or **catcodeshl**) record.
Any **code** can be used,. However, the use of **code = MECH** will generate a 'copy memory to VFO' command one second later with code = **MTOV**. If you don't need the 'copy after one second' feature, (eg: with Yaesu) then don't configure an **MTOV** record in catcodes (or catcodesciv or catcodeshl).

Memory Selector See examples in IC-7000 - memory channel selection
piWebCAT has a keypad to select memories which have been preprogrammed on the radio.



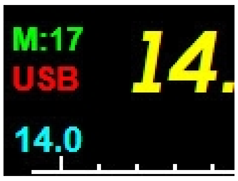
*The command must use **code = MPAD** - both in the **buttons** table and the **catcodes** table (or **buttonsciv** and **catcodesciv**).*

The **buttons** table entry has **code = MPAD** and **action = S** (single momentary action.)
This causes the button to display the keypad.
No data configuration is needed (ie von, voff etc)
The data will be the number of the button selection.

The **catcodes** table is configured with **code = MPAD** and for the radio's memory selector command.

When you click the button, a numeric keypad pops up as shown. You simply select the memory by number and click **OK**. The OK button sends an **MPAD** command to the server.

In the above example, I have configured a spare button **RR** for this job.
I have set **color** to navy and **caption** to MPad.
(Note that the popup keypad appears at a fixed initial position, but can then be dragged around the window.)



When a memory channel is selected using the keypad, a memory indicator appears to the left of the frequency display, eg: **M:17** as shown left.
The selected memory needs to *quickly* applied to the VFO
.. either automatically (Yaesu) or with the MOTV feature. This is because any consequential frequency change after three seconds will clear the indicator.

ASCII - eg: FTdx101D

For the FTdx101D, memory select is configured as **setmask = MChtu;**
(h, t and u map to hundreds, tens and units of the actual data to be sent = memory number).
The **MC** command selects the memory *AND applies it to the current VFO.*

Fixed channel buttons
You can program one or more buttons to select fixed memories.
Use **code = MECH** with one **buttons** record per button.
All buttons share a single **catcodes** record with **code = MECH** and configured with the **MC** command as above.
Only configure a **code = MTOV** command in **catcodes** (as described above) if you radio's memory select command
does not copy the selected memory to the VFO.

Using the memories keypad
Program one button with **caption = MPad** **code = MPAD** to launch the keypad.
Program a **catcodes** record with **code = MPAD** and for the **MC** command as above.

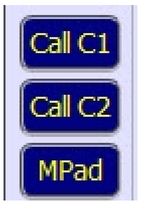
Icom CI-V: IC7000 and more modern radios

The IC7000 has five memory banks, A to E and 105 memory channels in each bank, 0001 to 0105.
In addition, channels 0106 and 0107 are call channels, C1 and C2.

Important: *data is sent as bcd (binary coded decimal)*
If Icom specify data as 0106 that means you must set ***datadigits = 4****, irrespective of the actual number.*
(0106 will be sent as hexadecimal bytes 0x01 0x06)

If I select a memory bank, A to E on the radio, then that selection is retained.
I can then use the memory channel CAT command to select the channel.
This is: command = 0x08 sub-command = channel number (or data = channel number on current Icom radios)

I configured three buttons:



See left - 3 spare buttons configured. Described below. Also in CI-V configuration examples
Two single channel buttons for channels 106 and 107 (= C1 and C2)
action = S, **vx = X**, **code = MECH** and **von = 106 / 107**

A single button to launch the memory selecting numeric keypad.
action = S, **vx = X**, **code = MPAD**. There is no data ... the data will be the number entered.

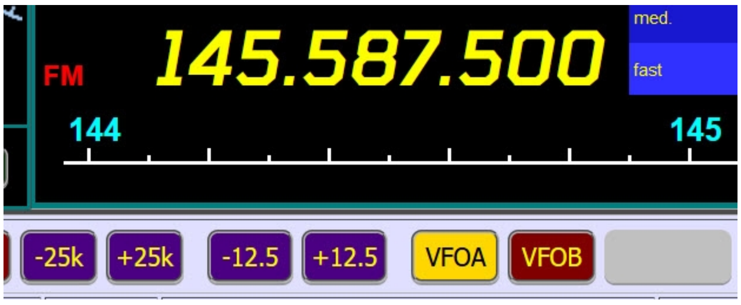
We need in **catcodesciv**
- a record with **code = MECH**, **cmdtype = C_DATA**, **command = 08**. **datadigits = 4**
- a record with **code = MPAD**, **cmdtype = C_DATA**, **command = 08**. **datadigits = 4**

Modern Icom radios
I looked at the Icom IC-7610 advanced manual. It has 100 memory channels with the same CAT command as the (15 yr old) IC7000. I could see no mention of memory banks A to E as used on the IC7000.
I have not provided an Icon memory bank selector in piWebCAT.

Note that the CI-V manuals for the two radios specify command 08 differently (but I think they are the same)
- IC-7000 - command = 08 subcommand = 0001 to 0105 no data cmdtype = C_DATA works ok.
- IC-7610 - command = 08 no subcommand data = 0001 to 0099 cmdtype = C_DATA should be ok

3.12 Frequency step up and down buttons

This feature allows you to program a button to shift the frequency of the current VFO up or down by a specified frequency step.



The user can program any button as a frequency Up ro Down button.
Here, I have four such buttons in my IC7000 Hamlib configuration. They are: +/- 25 kHz and +/- 12.5 kHz.

The first click moves the frequency to the next multiple of 12.5 kHz or 25 kHz. Thereafter, the buttons step at the specified interval.

On the first click, the frequency is initialised to a multiple of the step.
ie:

With a step of 12.5kHz and an initial frequency of 145.149902
- the first click of the **UP** button sets frequency = 145.150000
- and then 145.162500, 145.175000, 145.187500 etc

For **DOWN** button, we would have:
- 145.175085, 145.175000, 145.162500 etc

To set up UP and DOWN buttons, we only need **buttons** (or **buttonsciv**) records.

The button records must have **code = FRUP** or **code = FRDN**.
(FRUP and FRDN are hard coded in piWebCAT)

The frequency step in Hz is placed in the **von** field.

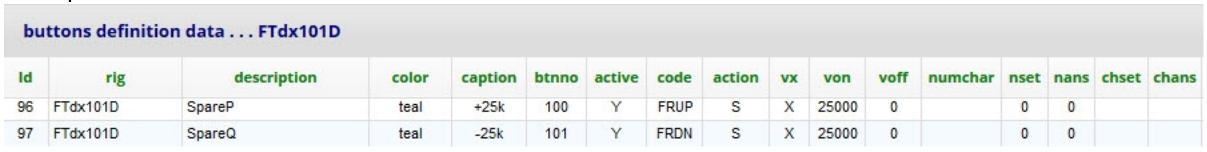
Clicking the buttons causes a tuning event using the standard tuning mechanism.
ie: as set up in **catcodes** (or **catcoedsciv**) with code = FREQ.

The buttons should be single action, ie: **action = S**.

Set **vx = X.** (piWebCAT will automatically send the commands for the current VFO)

Colour and caption a freely configurable as usual.

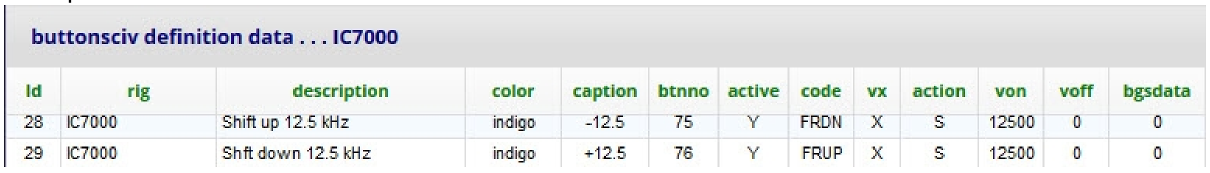
Examples for FTdx101D



buttons definition data . . . FTdx101D

Id	rig	description	color	caption	btnno	active	code	action	vx	von	voff	numchar	nset	nans	chset	chans
96	FTdx101D	SpareP	teal	+25k	100	Y	FRUP	S	X	25000	0		0	0		
97	FTdx101D	SpareQ	teal	-25k	101	Y	FRDN	S	X	25000	0		0	0		

Examples for IC-7000



buttonsciv definition data . . . IC7000

Id	rig	description	color	caption	btnno	active	code	vx	action	von	voff	bgsdata
28	IC7000	Shift up 12.5 kHz	indigo	-12.5	75	Y	FRDN	X	S	12500	0	0
29	IC7000	Shft down 12.5 kHz	indigo	+12.5	76	Y	FRUP	X	S	12500	0	0

3.13 MySQL, MySQL Front, HeidiSQL scripts

piWebCAT stores configuration data and your station log in a MariaDB (MySQL) database on the RPi.
The database is accessed:

- by piWebCAT's main control window (catcontrol.php) during normal operation.
- by piWebCAT's configuration and meter calibration editors.
- by external MySQL editor / toolkit applications via wired or wifi LAN.
  MySQL Front was used extensively in development.
  HeidiSQL is an alternative good database editor / toolkit which I have recently used.

*The following uses **MySQL Front**, but the processes are achievable using **HeidiSQL** or other tools.*

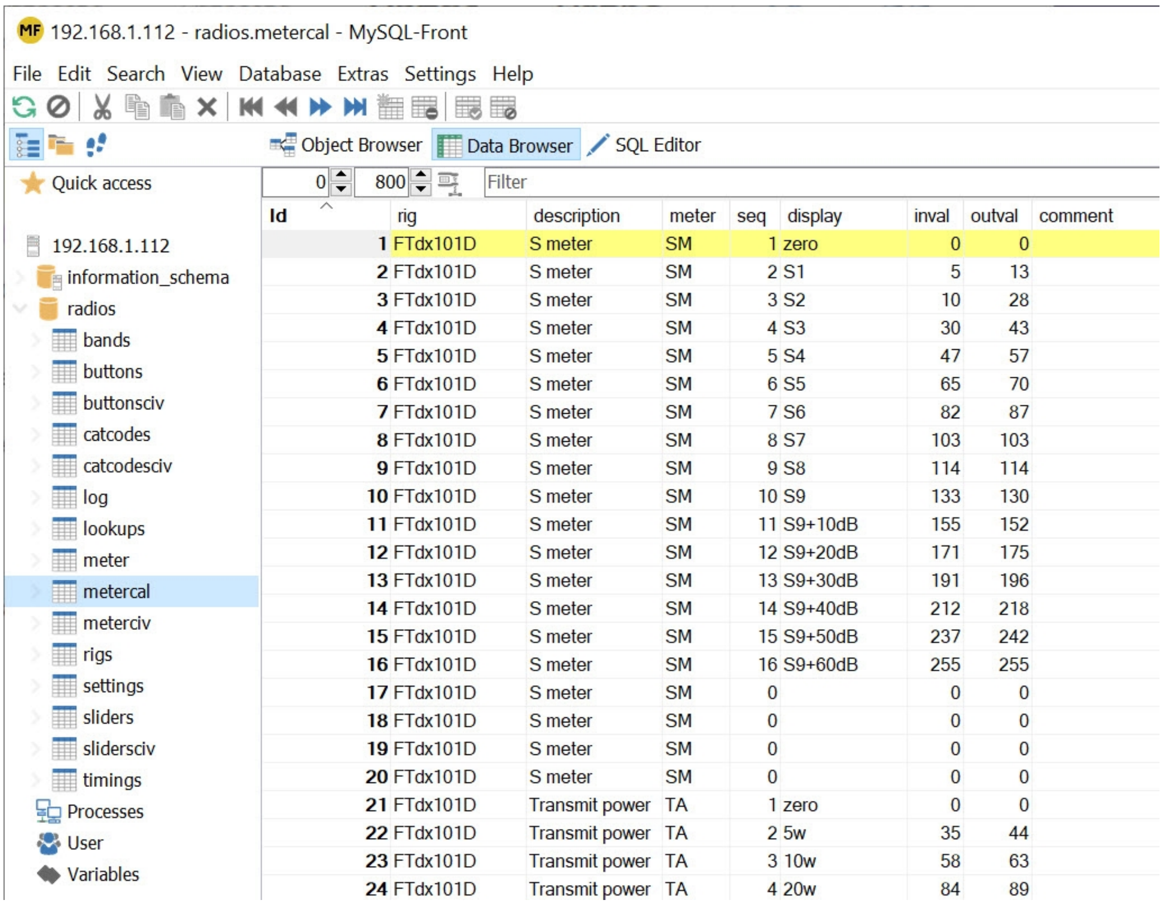
MySQL Front is a Windows front end for a MySQL database server.
It allows you to connect to database sources or import Text, SQL, MS Excel, MS Access, and ODBC files.
Dialog based data handling simplifies editing. SQL commands can also be run for automated editing tasks.

Please download and install MySQL Front on your PC. See 14.1 File downloads
After installation:
Use **File | Open connection** > **New**
Give it a name- why not piWebCAT. Access parameters are:

Host = **192.168.1.112** - or whatever else is the IP address of your RPi
Port = **3306 - always** user = **piwebcat** password = **feline** database = **radios**

MySQL Front should connect to the database.
Familiarise yourself with the program.
Note the object browser tab (table metadata definitions), data browser tab and SQL editor tab
Below shows the database (**radios**) expanded to show the list of tables and **data browsing** of **metercal.**

Editing is directly on the cells. Moving to another cell **saves** the edit.
New entries can be made by scrolling down beyond the bottom of the table.
The **Id** field can be changed to control table ordering, *but must not be duplicated.*
MySQL Front shows the whole table, whereas piWebCAT's editor shows only records for the selected radio.



Id	rig	description	meter	seq	display	inval	outval	comment
1	FTdx101D	S meter	SM	1	zero	0	0	
2	FTdx101D	S meter	SM	2	S1	5	13	
3	FTdx101D	S meter	SM	3	S2	10	28	
4	FTdx101D	S meter	SM	4	S3	30	43	
5	FTdx101D	S meter	SM	5	S4	47	57	
6	FTdx101D	S meter	SM	6	S5	65	70	
7	FTdx101D	S meter	SM	7	S6	82	87	
8	FTdx101D	S meter	SM	8	S7	103	103	
9	FTdx101D	S meter	SM	9	S8	114	114	
10	FTdx101D	S meter	SM	10	S9	133	130	
11	FTdx101D	S meter	SM	11	S9+10dB	155	152	
12	FTdx101D	S meter	SM	12	S9+20dB	171	175	
13	FTdx101D	S meter	SM	13	S9+30dB	191	196	
14	FTdx101D	S meter	SM	14	S9+40dB	212	218	
15	FTdx101D	S meter	SM	15	S9+50dB	237	242	
16	FTdx101D	S meter	SM	16	S9+60dB	255	255	
17	FTdx101D	S meter	SM	0		0	0	
18	FTdx101D	S meter	SM	0		0	0	
19	FTdx101D	S meter	SM	0		0	0	
20	FTdx101D	S meter	SM	0		0	0	
21	FTdx101D	Transmit power	TA	1	zero	0	0	
22	FTdx101D	Transmit power	TA	2	5w	35	44	
23	FTdx101D	Transmit power	TA	3	10w	58	63	
24	FTdx101D	Transmit power	TA	4	20w	84	89	

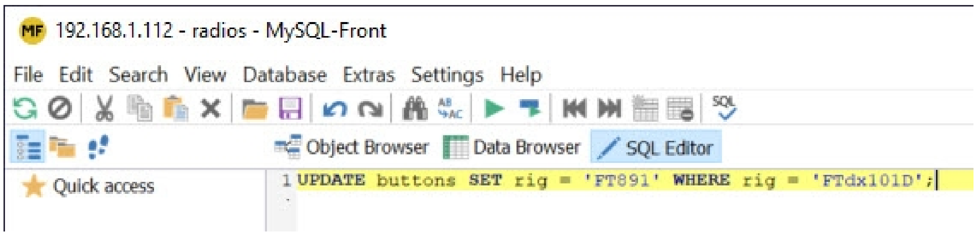
MySQL database query language

MySQL has a large number commands.
The few discussed here are relevant to managing piWebCAT data.

Below is a MySQL query in the Script editor tab.

Note that all queries must be terminated with a semicolon.

The script is executed by the ▶ button.   Refresh (⟳) is then needed on the modified table.


192.168.1.112 - radios - MySQL-Front
File Edit Search View Database Extras Settings Help
Object Browser
Data Browser
SQL Editor
Quick access

```
UPDATE buttons SET rig = 'FT891' WHERE rig = 'FTdx101D';
```

An example NOT to use: ~~`UPDATE buttons SET rig = 'FT891';`~~
This would set *every* record to rig = 'FT891'   ... which you do not want.
(Manual editing to reverse this would be difficult as you have lost the rig labeling.!!)

The example in the image above is:

```
UPDATE buttons SET rig = 'FT891' WHERE rig = 'FTdx101D';
```

This simply relabels all the the FTdx101D records as FT891 records.
The FTdx101D and FT891 CAT command systems are close to identical. (2m and 70cm need adding).
So if your rig is FT891:

- You can use piWebCAT as supplied pretending to be an FTdx101D.
- You can then relabel the records as FT891 if you wish.

You need to perform the process for buttons, sliders, catcodes, meter, timing, lookups, rigs and metercal,
AND then change the selected radio in settings to F891 (The drop down selector would now do this.)

Note that if you are dealing with **Icom** radios, then the tables are:
buttonsciv, sliderciv, catcodesciv, meterciv, timing, lookups, rigs and metercal,

Deleting records;

To delete all IC7000 records from the buttonsciv table we would use:

```
DELETE FROM buttonsciv WHERE rig = 'IC7000';
```

3.14 piWebCAT . MySQL Font, HeidiSQL - database backups

Please download and install **MySQL front** (free database editor)
(HeidiSQL is a good alternative.)

Use **File | Open connection** > **New**
Give it a name- why not piWebCAT

Host = 192.168.1.117 - or whatever else is the IP address of your RPi
Port = 3306 - always
user = piwebcat
password = feline
database = radios
.
It should connect to the database
Familiarise yourself with the program.
Note the object browser (table metadata definitions) and data browser tabs.

As supplied, the micro SD card image has configurations for:

- FTdx101D using direct piWebCAT (ASCII) text system
- FT847, FT818 and FT920 using the older (YAESU5) Yaesu 5 byte system.
- IC7000 using Icom CI-V
- A learning progression of Transceiver-H-A, -H-B and -H-C using Hamlib.
  These use --vfo mode for dual VFO systems (Working with my FTdx101D)
- A learning progression of Transceiver-H-A-NV, -H-B-NV and H-C-NV using Hamlib.
  These do not use --vfo mode. working with my Icom IC7000.
  Use with most Icom transceivers and other which do not provide CAT access to the background VFO.

Exporting the database

Highlight the database: **radios** in the left column
Right mouse click > export > to SQL file . choose folder filename > open
Choose **Structure** & **data** & **drop before create**.
You have just backed up the whole radios database content (tables) !!

To restore the backup in the future, highlight **radios** then R mouse - import > SQL file.
Warning: .. As soon as you click import, the import runs and overwrites your database.
(You are not asked to confirm .... it just happens!)

The **drop before create** option means that the SQL file will destroy all the tables before restoring them from file.

You now have a back up of my configuration (There is also one on the website!)
You can safely play!

SQL scripts

I provide some prewritten SQL scripts.
The most important is the one to duplicate a source radio to a new radio within the database.
This involves duplicating the source radio's records in each table.
The identifying **rig** field in the copy has the name of the new radio.
The unique **Id** fields in the records have new unique values.

3.15 MySQL command and scripts. MySQL Front

MySQL database access

Please download and install the database editor / manager application: **MySQL Front.**

Note that MySQL Front will, by default present all records in a database table.
ie: showing all records for all configured rigs. Sorting by rig name facilitates presentation by rig name.
piWebCat's editor only shows records for the selected rig.

I have extensively used MySQL Front but their are alternative database editor / toolkits.
HeidiSQL is a good alternative.

External database access uses:
host = **IP address** (eg 192.168.1.113) port = **3306**
user = **piwebcat** password = **feline** database = **radios**

Internal database access from piWebCAT is specified in webserver file:
/var/www/html/cat/phpfiles/wcmysql.php
and has the same user name and password. (host = localhost because internal access)

```
//  MYSQL access
$DbServer = 'localhost';
$DbUser = 'piwebcat';
$DbPw = 'feline';
$Database = 'radios';
```

MySQL scripts - these are listed in the the following section 3.15

MySQL database creation, configuration, data read and write and other operations all use text-based MySQL commands.
These are the same whether used internally by piWebCAT or for external access by MySQL Front or other SQL based programs.
External scripts often contain multiple command lines.
All commands must be terminated with semi-colon.

Some typical MySQL command lines are shown below.

```
UPDATE buttons SET rig = "FT2000" WHERE rig = "FTdx101D";
```
This, if applied to all relevant tables would be used to change he name of a rig configuration.
eg: when adapting my FTdx101D configuration for an FT2000.

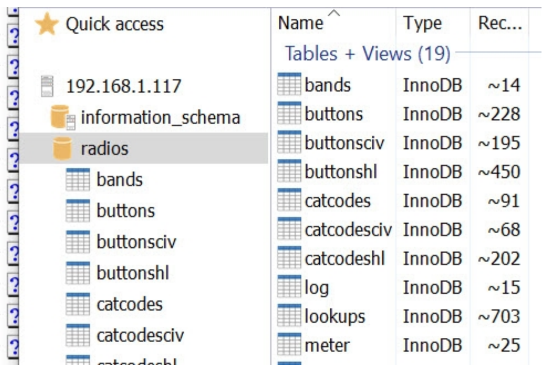
```
DELETE FROM catcodesdciv WHERE rig = "IC7000";
```
This if applied to all relevant tables would be used to remove IC7000 from the database.

```
UPDATE catcodeshl SET readmask = REPLACE(readmask, "Main","VFOA")
                                            WHERE rig = "FTdx101D-H";
```
This selectively acts on records in the **catcodeshl** table with rig = "Ftdx101D".
It changes all occurrences of "Main" in the readmask field to "VFOA".

```
DROP TABLE IF EXISTS `buttons`;
```
This will completely remove the table **buttons** thereby damaging the configuration for all rigs using table **buttons,** but not those using **buttonsciv** (Icom) or **buttonshl** (Hamlib connected).

Running scripts in MySQL Front.

On connecting to the radios database we have:

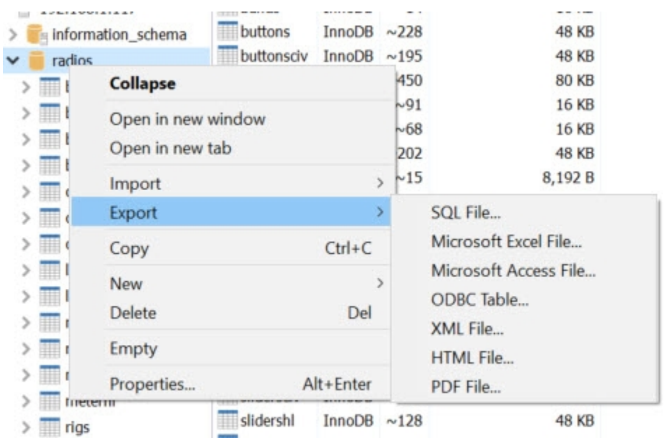


Name	Type	Rec...
bands	InnoDB	~14
buttons	InnoDB	~228
buttonsciv	InnoDB	~195
buttonshl	InnoDB	~450
catcodes	InnoDB	~91
catcodesciv	InnoDB	~68
catcodeshl	InnoDB	~202
log	InnoDB	~15
lookups	InnoDB	~703
meter	InnoDB	~25

Left mouse click on the database name (**radios**) displays the tables as shown.
Left mouse click on a table name will display its structure (headed: Object browser) or its data (headed Data browser).

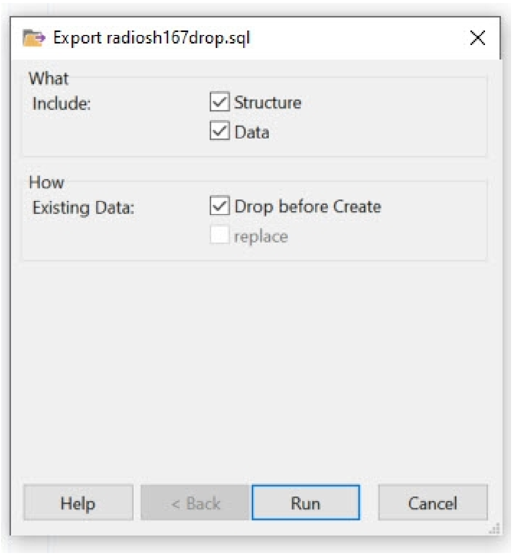
Right mouse click gives a list of options: Copy, New, Delete, Empty, Rename, Properties and also **Export** and **Import** which can be used for backup and restore.

Backing up the whole database



In the image left, I right-clicked on the database name: **radios** and the left-clicked export.
There are a number of export options.
The SQL option should be used for backup.

After selecting SQL you specify a filename and location and then finally there is the option box below.


Export radiosh167drop.sql

What
Include: [✓ Structure] [✓ Data]

How
Existing Data: [✓ Drop before Create] replace

These are export options.

The selected options are appropriate for **backup.**

If you subsequently import the exported SQL file, then the whole database (including your station log) will be deleted and the regenerated from the saved SQL file.

Selective table backup
If, for example, you wanted to preserve the station log file separately, then simply perform the above process starting with right-click on the **log** table name in the left column.
The resulting file can then be used to selectively restore the log.

Note that any table can be exported to an excel file from MySQL or from the piWebCAT editor.

Examine a backup script - Use Notepad

You will see that the section for each table consists of:

- **`DROP TABLE IF EXISTS 'buttons';`**
- **`CREATE TABLE 'buttons' ( ....`** followed by the table structure
- **`INSERT INTO TBALE 'buttons' (....`** followed by all the data

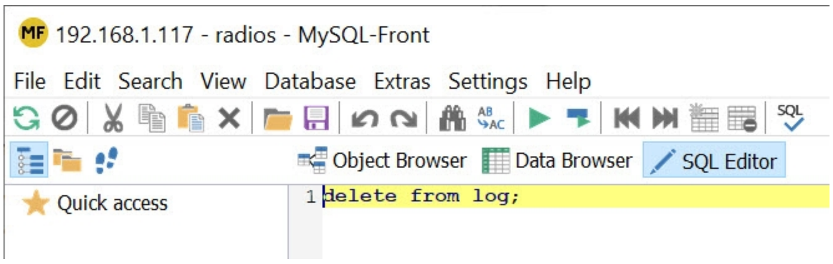
Using MySQL Front's SQL tab

MySQL Front has three tabs at the top of the page:

● Object browser	The structure (metadata) of the selected database table (R -mouse for properties, edit etc)
● Data browser	A spreadsheet-like presentation of the data - with editing capability
● SQL Editor	A page to enter individual MySQL commands (terminated in a semicolon) or multiple commands. The commands apply to the whole database (not just the selected table)

In the example below, a simple MySQL command has been typed: **`delete from log;`**
This clears data from the log table (but not the structure = metadata)
I use it to clear my log before generating an SD card for distribution.
However, before doing this I must save my log as follows:
  Expand the radios database on the left.
  R mouse click the log table item and then click Export - SQL file and specify a filename.

The script is executed by clicking the ▶ button.

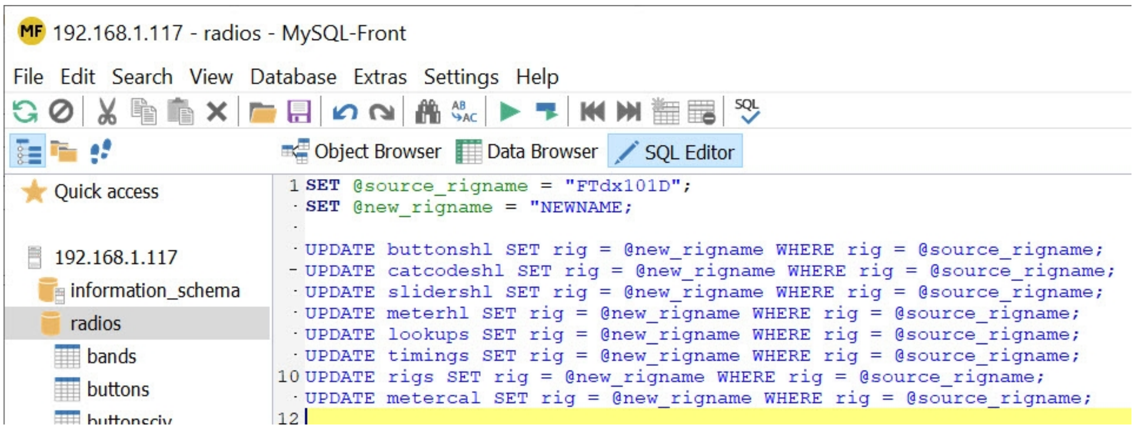

192.168.1.117 - radios - MySQL-Front
File Edit Search View Database Extras Settings Help
Object Browser
Data Browser
SQL Editor
Quick access

```
delete from log;
```

piWebCAT SQL script library (downloadable)

The SQL text below was inserted by R mouse - Paste from file.
The file used was : **Rename-rig_Hamlib.sql.**
This is in the small library of SQL file that I supply. It does what it says.
You need to edit the first two (SET) lines to contain the correct old and new rig names.
Note that in MySQL syntax, **@source_rigname** and and **@new_rigname** are variables that are loaded with the name and then are used in the eight UPDATE commands.

The script is executed by clicking the ▶ button. This runs all ten lines of code.



```
SET @source_rigname = "FTdx101D";
SET @new_rigname = "NEWNAME;

UPDATE buttonshl SET rig = @new_rigname WHERE rig = @source_rigname;
UPDATE catcodeshl SET rig = @new_rigname WHERE rig = @source_rigname;
UPDATE slidershl SET rig = @new_rigname WHERE rig = @source_rigname;
UPDATE meterhl SET rig = @new_rigname WHERE rig = @source_rigname;
UPDATE lookups SET rig = @new_rigname WHERE rig = @source_rigname;
UPDATE timings SET rig = @new_rigname WHERE rig = @source_rigname;
UPDATE rigs SET rig = @new_rigname WHERE rig = @source_rigname;
UPDATE metercal SET rig = @new_rigname WHERE rig = @source_rigname;
```

Adding new records

Configuration tables can be displayed and edited in piWebCAT's editor OR by MySQL Front.

piWebCAT's editors only show items in the table for the selected rig.
MySQL Front shows the whole table

Both editor systems show the **Id** field which is a unique identifier that cannot be duplicated.
The displays are sorted by default by **Id**. Changing the **Id** value (to a value that is not use) will, after a refresh reorder the table.
You can sometimes use this technique to control the new record's display position.
However, the tables in MySQL Front can be sorted on a column by clicking the column header (As in Excel)

Adding a record in MySQL Front.
Scroll to below the bottom of the table. A new record row opens up with the next sequential Id number assigned.
Enter data. SImply click elsewhere to save.
You may then be able to move the record by changing the Id field (but only to an Id value that is not is use)

Adding a new record in piWebCAT's editors is described in section 3.4 Editor page operation

3.16. Cloning radios in piWebCAT

To configure piWebCAT for your radio, you have three options:

- Add all the necessary database records 'manually' using the built in editor or MySQL Front.
  This is good learning process but a lot of work.
  There 90 buttons and 27 sliders. If you want the unused controls to be inactivated with a grayed out appearance, then you have to add a record for each one with: btnno=nn, active = N.
- Modify one of the radios in the SD card download.
- Clone one of the radios in the SD card download.

Cloning a radio **eg: IC7000 to NEWNAME** (another Icom rig)

Different Icom radios radios have similar sets of commands.
Some CAT codes will be the same. Many will be different.

Modifying a clone is much easier than creating a completely new radio configuration.

A small library of MySQL scripts is available. See next section : 3.17 Useful MySQL scripts

For each table (eg: buttonsciv) , cloning consists of:

- Copying all the records to a temporary table.
- Changing the rig field from IC7000 to NEWNAME.
- Deleting the Id field (primary key) because Id values must be unique when copied back to buttonsciv.
  (New, unique Id values will automatically be assigned on reinsertion on an auto-increment basis.)
- Also, I suggest setting the **metercal inval** and **outval** fields to 0. (indicates calibration not yet done)
- Finally, inserting the records from the temporary table into buttonsciv.

Two of the supplied scripts are shown pasted into MySQL Front in the following two pages.
The files are **Duplicate_rig_CIV.sql** and **Duplicate_rig_ASCII.sql.**
They are text files that can be edited using notepad or in the MySQL Front Script editor tab.

In order to use them, you must very carefully change the rig names to those of your choice.
(Please do a whole database backup before running them!! See MySQL Front backups)

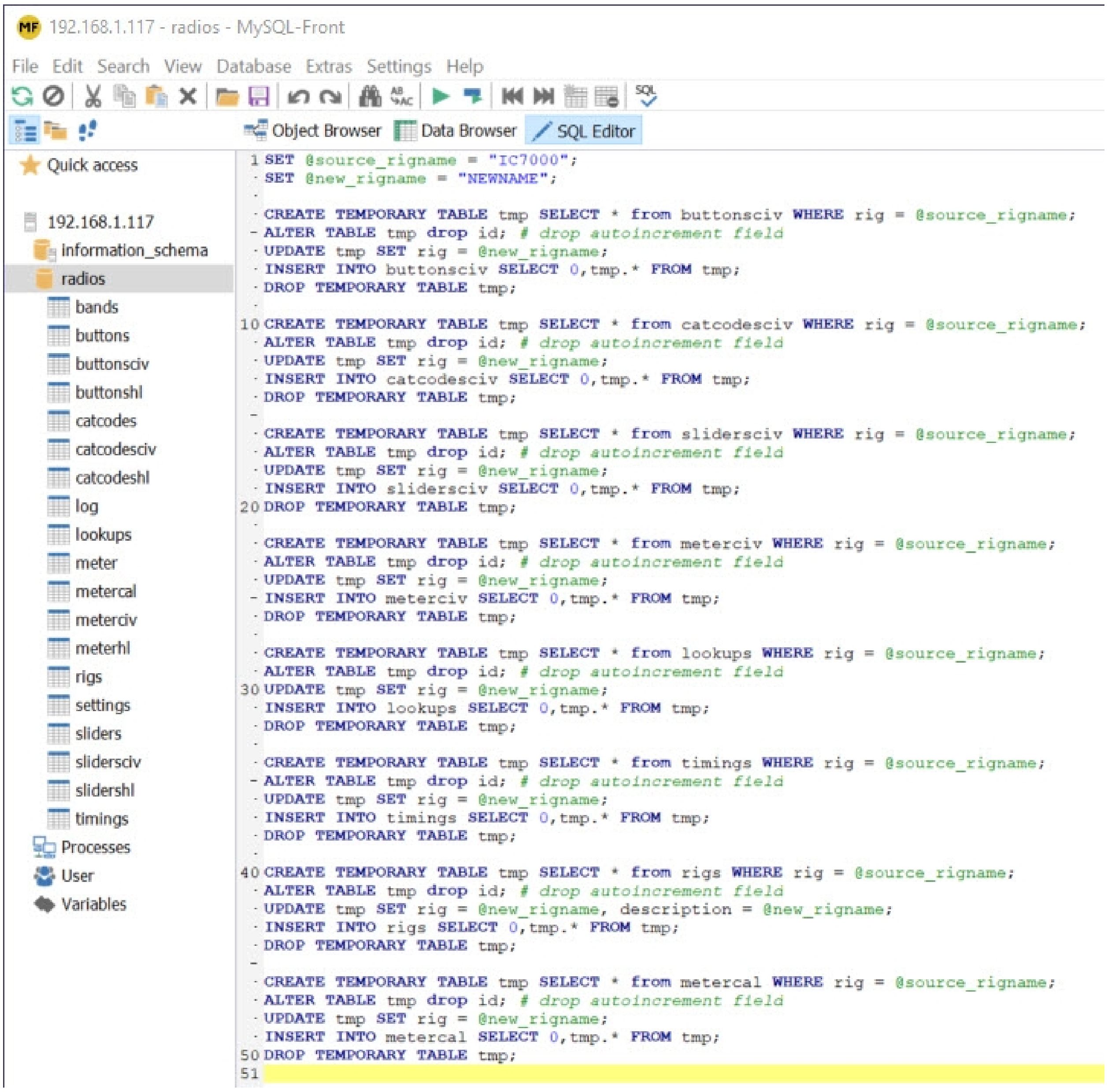
Note that these scripts are not specific to MySQL Front. They would run ok using any MySQL editor program.

MySQL Front with Duplicate_rig_CIV.sql loaded

The script was loaded by using Right mouse click > Paste from file
(Make sure that the window is clear of scripts before doing this .. otherwise you will run both scripts!

Clicking the ▶ button performs the clone in a fraction of a second.

Note that in MySQL syntax, **@source_rigname** and and **@new_rigname** are variables that are loaded with the name and then are used in the following commands.



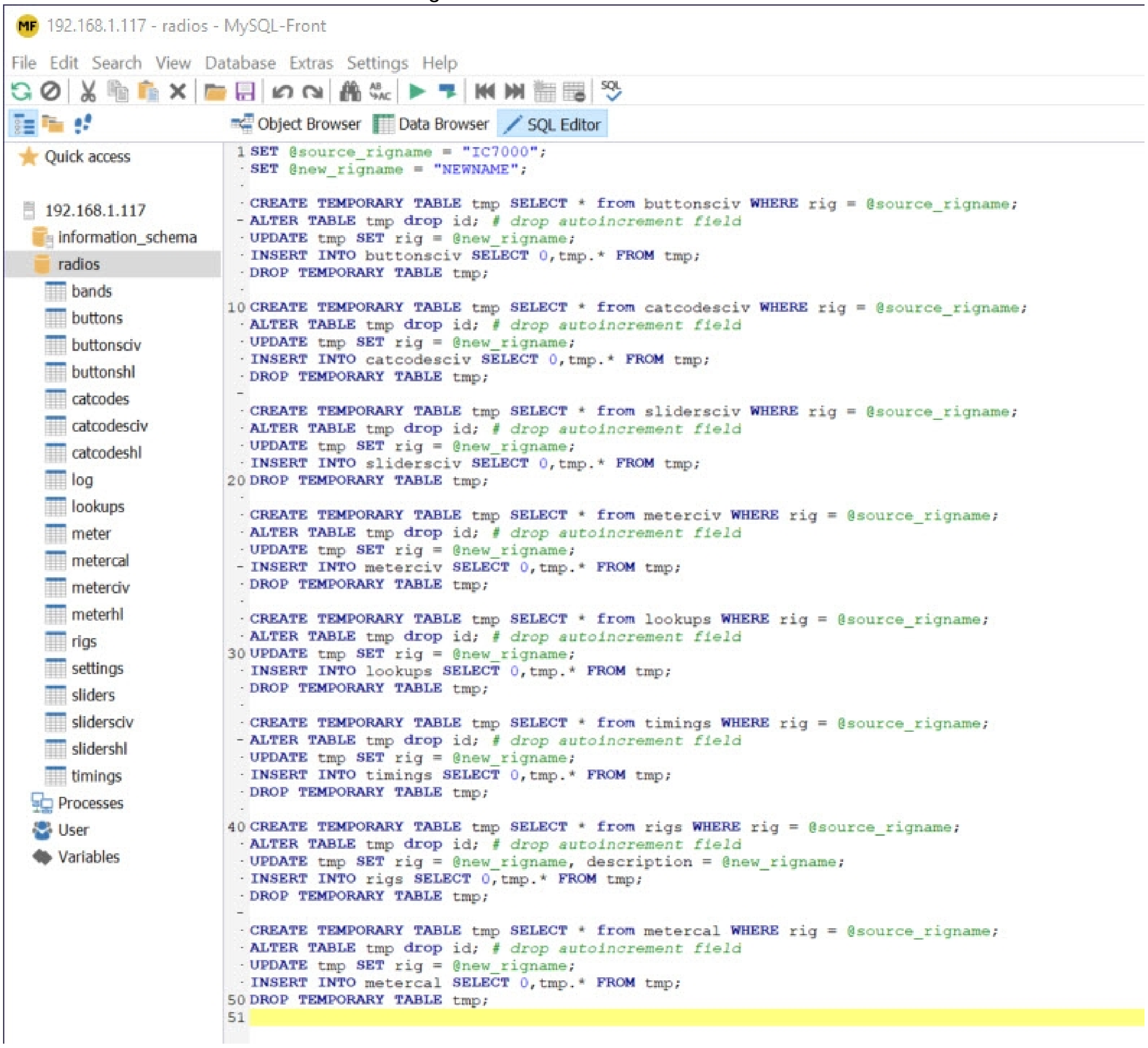
```
SET @source_rigname = "IC7000";
SET @new_rigname = "NEWNAME";

CREATE TEMPORARY TABLE tmp SELECT * from buttonsciv WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO buttonsciv SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from catcodesciv WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO catcodesciv SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from slidersciv WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO slidersciv SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from meterciv WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO meterciv SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from lookups WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO lookups SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from timings WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO timings SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from rigs WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname, description = @new_rigname;
INSERT INTO rigs SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from metercal WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO metercal SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;
```

MySQL Front with Duplicate_rig_Hamlib.sql loaded

The script was loaded by using Right mouse click > Paste from file
(Make sure that the window is clear of scripts before doing this .. otherwise you will run both scripts!

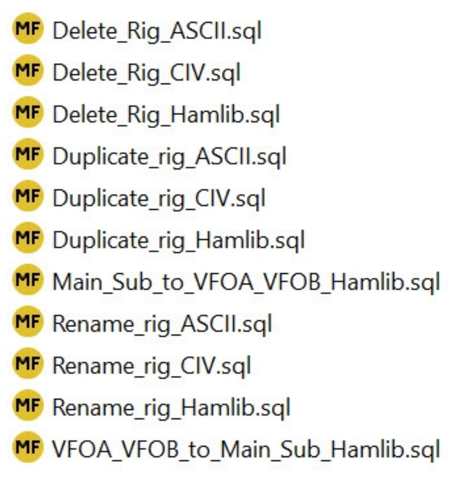
Clicking the button performs the clone in a fraction of a second.

Note that in MySQL syntax, @**source_rigname** and and @**new_rigname** are variables that are loaded with the name and then are used in the following commands.



```
SET @source_rigname = "IC7000";
SET @new_rigname = "NEWNAME";

CREATE TEMPORARY TABLE tmp SELECT * from buttonsciv WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO buttonsciv SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from catcodesciv WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO catcodesciv SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from slidersciv WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO slidersciv SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from meterciv WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO meterciv SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from lookups WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO lookups SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from timings WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO timings SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from rigs WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname, description = @new_rigname;
INSERT INTO rigs SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;

CREATE TEMPORARY TABLE tmp SELECT * from metercal WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO metercal SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;
```

3.17 Useful MySQL scripts provided with piWebCAT


Delete_Rig_ASCII.sql
Delete_Rig_CIV.sql
Delete_Rig_Hamlib.sql
Duplicate_rig_ASCII.sql
Duplicate_rig_CIV.sql
Duplicate_rig_Hamlib.sql
Main_Sub_to_VFOA_VFOB_Hamlib.sql
Rename_rig_ASCII.sql
Rename_rig_CIV.sql
Rename_rig_Hamlib.sql
VFOA_VFOB_to_Main_Sub_Hamlib.sql

I have provided a suite of SQL scripts to aid database data manipulation. Their names are listed left.

They are downloadable from the website as a zipped folder.
See: http://piwebcat.g3vpx.net/files/LibrarySQL.zip

They are also in folder: **/home/pi/LibrarySQL** on the SD card.
(Download with FileZilla for use on PC with MySQL Front.)

They need editing before use to set rig names etc.

The scripts are run in MySQL Front by right-click on the radios database item and then Import > SQL file.

A prior database backup is strongly suggested.

Example:
Duplicate_rig_Hamlib.sql

The start of this script is shown below.

```
SET @source_rigname = "FTdx101D";
SET @new_rigname = "NEWNAME;

CREATE TEMPORARY TABLE tmp SELECT * from buttonshl WHERE rig = @source_rigname;
ALTER TABLE tmp drop id; # drop autoincrement field
UPDATE tmp SET rig = @new_rigname;
INSERT INTO buttonshl SELECT 0,tmp.* FROM tmp;
DROP TEMPORARY TABLE tmp;
```

`CREATE TEMPORARY TABLE ...` is repeated for seven more tables

`@source_rigname` and **`@new_rigname`** are MySQL variables.

You need to substitute the name (**rig** field) of the rig you are duplicating for my FTdx101D
and your new rig name (**rig** field) for my NEWNAME.

Then save the script.

Then run MySQL Front on the database.
Right click the **radios** database item, top left. Then **Import**, then **SQL** and then select the script file and run it.

VFOA / Main etc scripts are provided because modern Yaesu rigs use Main / Sub rather than VFOA / VFOB and this is reflected in the command syntax of Hamlib.
So, if you want to use my FTdx101D-H configuration as the basis for rigs that use VFOA / VFOB in Hamlib then you need make the changes.

3.18 Updates with FTP - users and passwords

FTP access Host = RPi IP address (eg: 192.168.1.113) port = 21

Two users are installed on the SD card. Both have password = **feline.**

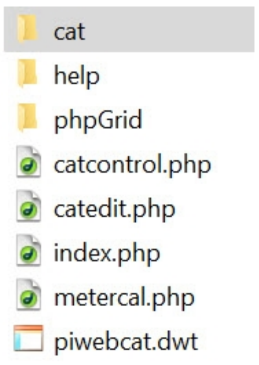
- user = **upload** This gives FTP access to the RPi webserver root: **/var/www/html**
- user = **piuser** This gives FTP access to the RPI directory: **/home/pi**

I use FileZilla FTP client for file transfer.
However, much of my webserver upload is direct from the Microsoft Expression web developer application (free download) where I configure the same FTP credentials.

Web server 'website structure' and updates

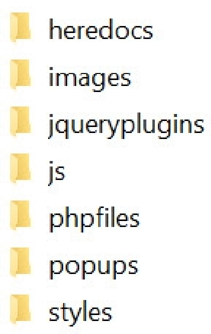
The website structure on the RPi is shown below in screen images of it in Windows on a PC.

Web root folder



cat
help
phpGrid
catcontrol.php
catedit.php
index.php
metercal.php
piwebcat.dwt

cat subfolder



heredocs
images
jqueryplugins
js
phpfiles
popups
styles

The website root has:

- Four **.php files** which are the three working web pages and the opening screen: index.php.
- Folder **phpGrid** (21Mbytes) which contains the imported phpGrid system for configuration editor grid generation.
- A **help** folder(22 Mbytes) containing this complete website as a built in help facility.
- The **cat** folder (0.71 Mbytes) which contains the rest of my code in subfolders.
- **piwebcat.dwt** is a Microsoft Expression4 template file for the three main web pages.

As can be seen from the above text, the phpGrid and help folders total 43 Mbytes, whereas the rest of the structure is only 0.82 Mbytes.
Therefore, some updates will not include the phpGrid and help folders.
The help and phpGrid folders need to be preserved if not in the update.

3.19 Javascript memory leaks - Web browser choice.

This section discusses the issue of web browser javascript memory leaks.
It arises because piWebCAT is making up to twenty client <> server transactions per second
(Many web browser applications sit there doing nothing until you use the keyboard or mouse)

My primary concern was that piWebCAT should be able to run for 2 hours or more without slowing enough to need a restart. This is achieved.
(A restart is by clicking the Control button .... and only takes about five seconds)
The situation varies between web browsers. Chrome on PC and on Android are good.

Memory leaks

Computer programs need to dynamically make recurring temporary usage of memory.
The memory reserve is often referred to as the **heap**.
Some of this **heap** memory is used for variables which are defined and used **locally** within a particular **function**. Memory is allocated from the heap for the duration of the function and then returned to the heap when the function exits. (jargon: procedure = function = subroutine)

A **memory leak** is when some of the allocated memory is not returned to the heap and therefore remains in an unused and inaccessible state.
Repetitive calling of a leaky procedure can result in a in growing unusable section of the heap.

Microprocessors
Small microprocessors often have very limited RAM memory
(eg: the ATxmega192A3 processor in my EncoderCAT project has only 16k bytes.
When programmed in a language such as C, complete avoidance of an memory leak is essential and fortunately easy to achieve.

Web browsers - javascipt
Javscript is the programming language that runs on web browser (ie: web client)
Simply googling 'javascipt memory leaks' reveals that:
- memory leaks are a common problem
- they are somewhat browser dependant.
- they are often not completely suppressed.

A **javascipt** memory leak and other issues can gradually slow down performance and even end up with the screen image fragmenting.

piWebCAT and memory leaks

I become aware of potential memory leak issues very late in the development, because:

- During most of the development, it was unusual to leave piWebCAT running for long periods:
  - the development process is very much a repetitive cycle of ***modify - run - observe*** rather than staring at the browser for long periods!
- Late in development, I increased the client <> server command rate (up to 20Hz) which potentially increases any memory leakage rate.

The memory leakage issue in piWebCAT arises because it is making up to 20 client <> server transactions per second.... whereas most web browser applications just sit there doing nothing until you press a key !!

One important cause of memory leakage in javascript is *defining **global variables** within a function*.
Javascipt will let you do this.
A **global variable** is a data storage item that is available throughout the javascipt code.
It is said to have **global scope**. You can set or read it's value from inside any function in the whole of the web page javascript code.

A global variable, (eg: freqMain) should not be defined inside a a function.
If it is defined within a function, then a new **instance** of it will be created every time that function is called.
Some functions in piWebCAT are called 10 or 20 times per second.

Variables that are only for use with in function should be defined within the function as: **var variableName**,
Memory is then grabbed from the heap when the function is called and released when the function exits..

I discovered late in the development the insertion use of **"use strict";** at the start of each program module.
This makes illegal the declaration of globals within a function. The system fails to run. The web browser debugging facility then locates the offending items.

I carefully removed all such items and also other identifiable known causes of memory leaks.
However, some gradual memory leakage remains.
An internet search for help on this reveals that I am not alone in not completely removing leaks.

Web browser choice:
At the time of writing, acceptable duration of operation depends on web browser choice.

- Firefox (and Firefox developer) can slow down unacceptably and fail in under two hours.
- Chrome for Windows and Android are much better choices in this respect.

However, Firefox has the advantage of being able to use the mouse thumbwheel for fine slider adjustment.
This is excellent for an RIT control.   So far, I have not achieved this feature with other browsers.

Javascript's heap memory management is said to use a so called **'garbage collection'** system.
This means that discarded heap memory is not restored to available heap on termination of the function that uses it. The garbage collection system periodically collects unused memory and restores it to the heap.
The system is a web browser facility and appears to have quite different performance on different browsers.

eg: Running piWebCAT over 1.5 hours, Firefox developer increased its stated heap size by 70 Mbytes whereas Chrome reported less than 1.5 Mbytes.

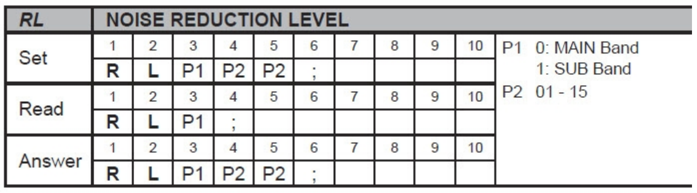
4.1 Command masks on ASCII radios

The examples are from the FTdx101D. Other ***modern*** Yaesu radios have a similar CAT system.
My configuration system for older Yaesu (5 - byte command) radios also uses these command masks.

piWebCAT's control actions are NOT read-modify-write (unlike my EncoderCAT project)
The controls are synchronised to the radio's settings at startup, on band change and by the reload button.
*This uses the **readmask** and **answermask** settings to request the data and interpret the answer.*
*Subsequent button or slider action commands use **setmask** with the new position of a slider,*
*or from known buttons states. ie: only the **set** command is used.*
(eg: for toggled buttons, piWebCAT uses the button's remembered on or off state to determine the new state.)

A example from the FTdx101D CAT manual - Noise Reduction Level.

The CAT manual entry is below. There are two identical receivers, **MAIN** and **SUB.**
(Note that the terminating semicolon is part of the command)



RL	NOISE REDUCTION LEVEL										
Set	1	2	3	4	5	6	7	8	9	10	P1 0: MAIN Band
	R	L	P1	P2	P2	;					1: SUB Band
Read	1	2	3	4	5	6	7	8	9	10	P2 01 - 15
	R	L	P1	;							
Answer	1	2	3	4	5	6	7	8	9	10	
	R	L	P1	P2	P2	;					

- **MAIN** - read = "RL0;" answer = "RL006;" set = "RL007;"
- **SUB** - read = "RL1;" answer = "RL106;" set = "RL107;"

In **MAIN.Answer** above: "**RL0**" is the command and "'**06**' is the variable data.

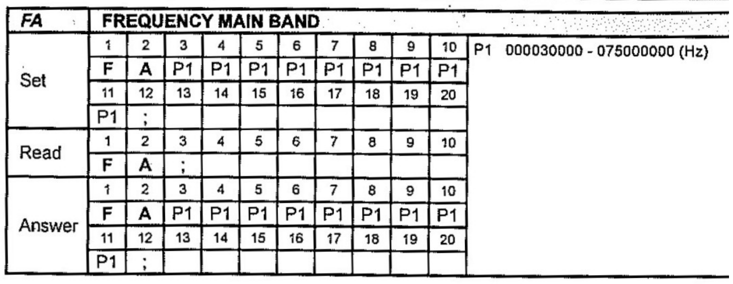
We configure the masks as follows::
readmask is "RL0;" for VfoA and RL1; fro VfoB
setmask and **answermask** are both RL0tu; for VfoA and RL1tu; for VfoB.

- "**RL0**" and RL1 are **fixed** as the first three characters.
- "'**tu**" indicates that the fourth and fifth characters are variable **tens** and **units**.

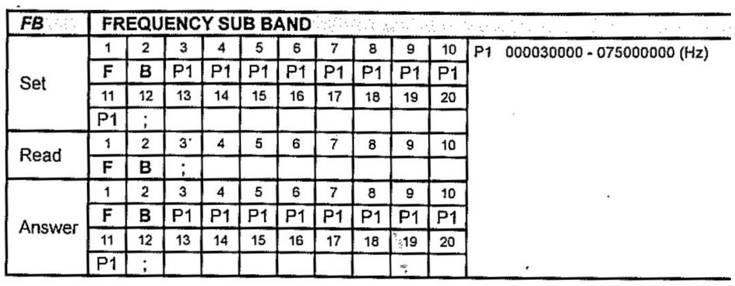
readbytes is 4 (ie: RL0; is 4 bytes) **setbytes** and **answerbytes** are 6 (ie: RL0tu; is six bytes)

Further example: frequency read and write

In piWebCAT, we tune with the current VFO which may be A or B
Yaesu CAT manual states



FA	FREQUENCY MAIN BAND										
Set	1	2	3	4	5	6	7	8	9	10	P1 000030000 - 075000000 (Hz)
	F	A	P1	P1	P1	P1	P1	P1	P1	P1	
	11	12	13	14	15	16	17	18	19	20	
	P1	;									
Read	1	2	3	4	5	6	7	8	9	10	
	F	A	;								
Answer	1	2	3	4	5	6	7	8	9	10	
	F	A	P1	P1	P1	P1	P1	P1	P1	P1	
	11	12	13	14	15	16	17	18	19	20	
	P1	;									



FB	FREQUENCY SUB BAND										
Set	1	2	3	4	5	6	7	8	9	10	P1 000030000 - 075000000 (Hz)
	F	B	P1	P1	P1	P1	P1	P1	P1	P1	
	11	12	13	14	15	16	17	18	19	20	
	P1	;									
Read	1	2	3	4	5	6	7	8	9	10	
	F	B	;								
Answer	1	2	3	4	5	6	7	8	9	10	
	F	B	P1	P1	P1	P1	P1	P1	P1	P1	
	11	12	13	14	15	16	17	18	19	20	
	P1	;									

For VFOA **readmask** = **FA;** **setmask** and **answermask** are **FAgfedcmhtu;**
numbers 0 - 999 999 999 ie: 0 - 999 Mhz
For VFOB **readmask = FB;** **setmask** and **answermask** = **FBgfedcmhtu;**
readbytes is 3 (ie: **FA;** is 3bytes) **setbytes** and **answerbytes** are both 12.

List of 'variable' characters

I have defined a small list of lower case characters that represent variable data.
All other characters outside this list are interpreted as **fixed.**

The characters are g, f, e, d, c, m, h, t, u (u = units, t= tens, h = hundreds etc)
There is also s = sign - representing '-' or '+'
$ is don't care

The codes are detailed below in an extract from EncoderCAT C source code:

EncoderCAT and piWebCAT use the same codes for character interpretation:

```
For decimal digit ascii characters, ie:  '0'  to  '9'
  switch(mb)
  {
    case 'g': mult = 100000000; break; // g = hundred millions
    case 'f': mult = 10000000; break;  // f = ten millions
    case 'e': mult = 1000000; break;   // e = millions
    case 'd': mult = 100000; break;           // d = hundred thousands
    case 'c': mult = 10000; break;            // c = ten thousands
    case 'm': mult = 1000; break;             // m = thousands
    case 'h': mult = 100; break;       // h = hundreds
    case 't': mult = 10; break;        // t = tens
    case 'u': mult = 1; break;                // u = units
  }


    // Also    s represents sign '-' or '+'
```

There is also **x** which is used for hexdecimal characters. eg: **xx** or **xxxx**
There are some hexadecimal code in modern Yaesu radios.
The main use is in the older Yaesu 5-byte command radios for which piWebCAT translates a ten-character configuration string into five hexadecimal bytes for transmission to the radio.

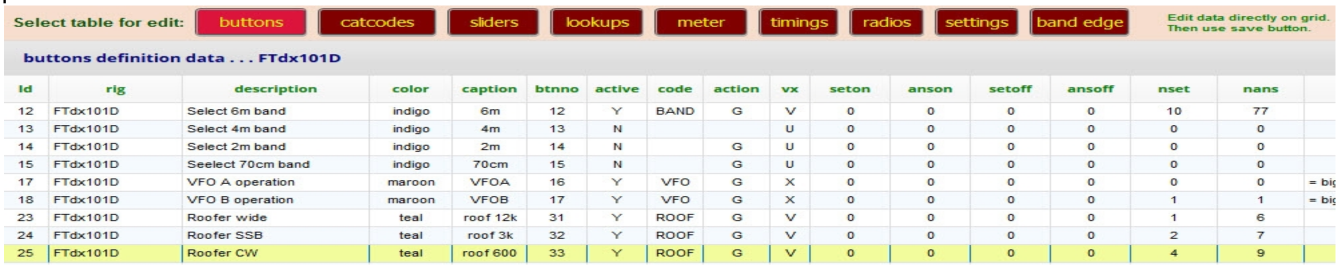
See Yaesu 5-byte

4.2 piWebCAT - Database table - buttons

See also buttons configuration notes   timing.disable

- **rig** The current radio - drop down selector (from **radios** table)
  Must have same spelling through the tables.
- **description** Descriptive text- no function
- **color** Button's background colour at startup. (or bright 'ON ' colour for a read-only LED button)
  Can be a standard HTML color, eg teal, indigo etc or a numeric colour:
  ie: #RRGGBB eg #223344 where 0x22 is red level (0x00 - 0xFF)
- **caption** The caption to be applied to the button at start up. Must fit the button's width.
  Try something and then observe. (Lower case letters are MUCH narrower!)
- **bttno** The button's unique, fixed, numeric identifier. See: Button and slider numbering
- **active** Y = button active N = button inactive S = active + sync (repetitive state update from rig)
  L = sync and 'LED' read-only indicator lamps (See: LED buttons)
- **code** 3 or 4 upper case characters. This links a button command to its action
  on the server. It must match the **code** for the linked record in **catcodes**.
  See below for more explanation.
- **vx** **V** for client to send A or B according to current VFO. (A and B catcodes records)
  X to send X (single non-VFO dependent server record. **U** no action to server.
  (U = do not participate in buttons state store and retrieve) See also: vx and abx
- **action** **S** = single momentary. Button flashes briefly. Sends **von** column value.
  T = toggled. Alternates on/off. Client code highlights it when on and remembers
  its current state. Sends **von** or **voff** column value.
  G = grouped. Is in a group of other **G** buttons *with the same **code**.*
  Only one of the group is highlighed. Each button in the group sends its data
  from the **nset/nans** columns or **chset/chans** columns- see below.
  M = meter button. **R** = slider reset button. **U** = unused.
- **seton** ON value for an on/off (toggling) button ie: action = **S** or **T** (usually 1)
- **setoff** OFF value for an on/off button ie: action = **S** or **T** (usually 0)
- **anson** ON value to match a button state command received from the radio.
- **ansoff** OFF value to match a button state command received from the radio.
- **nset** The send value for a button that is in a button **group**
- **nans** The answer matching value for a button in a **group** (not always the same as sent)
  See also end of section 3.7 - button group problem
- **bgsdata** Unused - Icom CIV only ... included to simplify RS232 / CIV compatibility.

The **button** table fields have to be set to match with the corresponding record(s) in the **catcodes** table.
piWebCAT's buttons table editor is shown below:



buttons definition data . . . FTdx101D

Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset	nans	
12	FTdx101D	Select 6m band	indigo	6m	12	Y	BAND	G	V	0	0	0	0	10	77	
13	FTdx101D	Select 4m band	indigo	4m	13	N			U	0	0	0	0	0	0	
14	FTdx101D	Select 2m band	indigo	2m	14	N		G	U	0	0	0	0	0	0	
15	FTdx101D	Seelect 70cm band	indigo	70cm	15	N		G	U	0	0	0	0	0	0	
17	FTdx101D	VFO A operation	maroon	VFOA	16	Y	VFO	G	X	0	0	0	0	0	0	= big
18	FTdx101D	VFO B operation	maroon	VFOB	17	Y	VFO	G	X	0	0	0	0	1	1	= big
23	FTdx101D	Roofer wide	teal	roof 12k	31	Y	ROOF	G	V	0	0	0	0	1	6	
24	FTdx101D	Roofer SSB	teal	roof 3k	32	Y	ROOF	G	V	0	0	0	0	2	7	
25	FTdx101D	Roofer CW	teal	roof 600	33	Y	ROOF	G	V	0	0	0	0	4	9	

The numeric fields are actually all nine-character string fields and so can accept hexadecimal characters A - F.

Unmatched button conditions

If a button group does not have a button to match all corresponding states on the rig, then an error arises.
My FTdx101D has fifteen modes. If I only configure buttons for seven, then selecting one of the other eight on the rig will cause a problem. This is discussed at the end of section 2.11 Learning guide Transceiver-H-A

Example (FTdx101D) - Preamp / IPO control

The CAT manual gives the command as PA0; to read the main Rx and PA1; to read the sub Rx status.
The set and answer data is PA0u; or PA1u; where u is a single ascii digit for the data.

The data to send (or read on startup) is IPO = 0 Amp 1 = 1 Amp 2 = 2

We have three buttons entries in the **buttons** table, one for each button.

We have two entries in the **catcodes** table, one for the Main receiver and and one for the Sub receiver.

All five carry the same code..... I have used PAMP
(This is not internally fixed ... so I could have used PRMP or PREA)

In the **buttons** table, all three buttons must have action = **G** (grouped)

We must set **vx = V** See: vx and abx
This makes the client send to the server the current VFO selection (A or B)
It also make client store the latest VFO A and VFO B settings for application on VFO change.

For each button the data is in the **nset** and **nans** fields and **numchar** = N
ie IPO button = 0 Amp1 button = 1 Amp 2 button = 2

When the Amp1 buttons is clicked, with VFO A is selected, the client sends to the server:
code = PAMP jobdata = 1 VFO = A

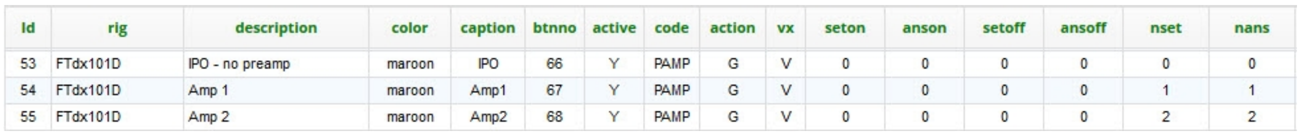
The server code pulls from the database **catcodes** table the record with:
code = PAMP **abx** = A

From this record the code picks up **sendmask** = PA0u; and **sendbytes** = 5.

It substitutes the jobdata of 1 for u in the mask and sends the message as **PA01;**

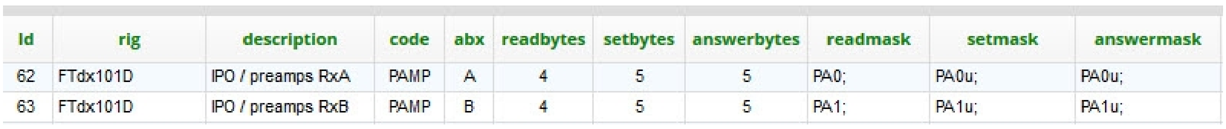
The table entries are shown below:
There are three **buttons** records for three groups buttons.
There are two **catcodes** records for two VFOs which have different commands

buttons



Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset	nans
53	FTdx101D	IPO - no preamp	maroon	IPO	66	Y	PAMP	G	V	0	0	0	0	0	0
54	FTdx101D	Amp 1	maroon	Amp1	67	Y	PAMP	G	V	0	0	0	0	1	1
55	FTdx101D	Amp 2	maroon	Amp2	68	Y	PAMP	G	V	0	0	0	0	2	2

catcodes



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
62	FTdx101D	IPO / preamps RxA	PAMP	A	4	5	5	PA0;	PA0u;	PA0u;
63	FTdx101D	IPO / preamps RxB	PAMP	B	4	5	5	PA1;	PA1u;	PA1u;

Tx meter buttons

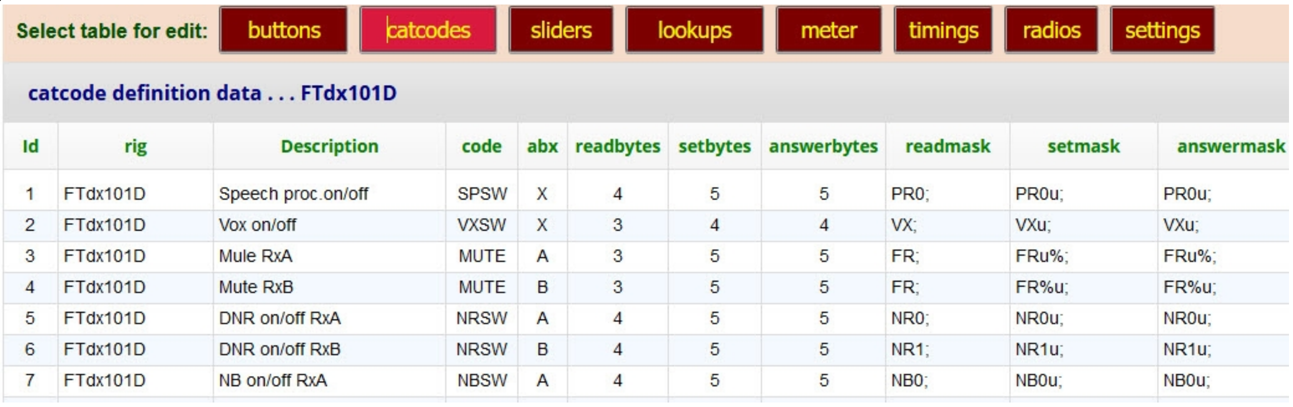
These do not communicate with the server - they select the Tx meter on the client.
action = M and the code field is fixed as **TXME**
Clicking the button does not issue a command to the radio. It records the Tx meter selection on the client so that the repetitive meter read process reads the correct meter. It also identifies the group in order to clear the group before highlighting the clicked button.
Meter reading codes are set up in the **meter** table.
The link to the meter table is the button numbers 61 to 65. See **meter** table section

4.3 piWebCAT - Database table - catcodes

catcodes is used on the server to communicate with the radio for buttons and frequency.
(None of its data is loaded to the client)

- **rig** The current radio - drop down selector (from **radios** table)
  Must have same spelling through the tables.
- **description** Descriptive text- no function
- **code** The link to the buttons table - use my defaults where possible.
  *Note that for frequency reading and setting, the code must be **FREQ*** (hard coded)
- **abx** A or B if there is pair of entries one for each VFO (eg: Mute RxA, RxB)
  X if A or B not relevant. (eg:swap VFOs, Tuner etc) See also: vx and abx
- **readbytes** The number of bytes in a read command (= no of chars in readmask)
- **setbytes** The number of bytes in a set command (= no of chars in setmask)
- **answerbytes** The number of bytes in an answer (= no of chars in answermask)
- **readmask** The character pattern of the read command, see Command masks
- **setmask** The character pattern of the set command. see Command masks
- **answermask** The character pattern of the answer. see Command masks

piWebCAT editor for **catcodes** is shown below:



catcode definition data . . . FTdx101D

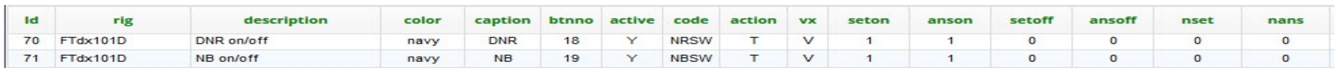
Id	rig	Description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
1	FTdx101D	Speech proc.on/off	SPSW	X	4	5	5	PR0;	PR0u;	PR0u;
2	FTdx101D	Vox on/off	VXSW	X	3	4	4	VX;	VXu;	VXu;
3	FTdx101D	Mule RxA	MUTE	A	3	5	5	FR;	FRu%;	FRu%;
4	FTdx101D	Mute RxB	MUTE	B	3	5	5	FR;	FR%u;	FR%u;
5	FTdx101D	DNR on/off RxA	NRSW	A	4	5	5	NR0;	NR0u;	NR0u;
6	FTdx101D	DNR on/off RxB	NRSW	B	4	5	5	NR1;	NR1u;	NR1u;
7	FTdx101D	NB on/off RxA	NBSW	A	4	5	5	NB0;	NB0u;	NB0u;

Note the **abx** column:
If there are separate commands for VfoA and VfoB, then there are two records with **abx** = A and **abx** = B.
Thus DNR on/off has two entries and speech proc on/off has one entry (so **abx** = X) See also: vx and abx

Examples:

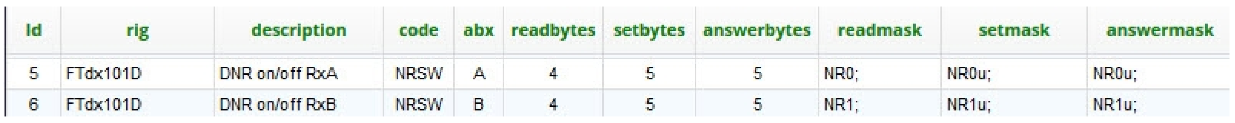
DNR on/off *This has one record in the **buttons** table and two records in **catcodes***
There is single button in the **buttons** table with **code** = NRSW, **von** = 1, **voff** = 0.
We set **vx** = V in **buttons** because the radios holds separate DNR settings for VFOs A and B.
The URL of a set message to the server includes: &code=NRSW&jobdata=1 (1 is von)
There are two entries in the **catcodes** table:
abx = A setmask = NR0u; and **abx** = B **setmask = NR1u;**
A or B will be sent to the radios according to piWebCAT's current VFO selection.
The u = units = single character of data. u will be '1' for on and '0' for off

FTdx101D configuration shown below
buttons



Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset	nans
70	FTdx101D	DNR on/off	navy	DNR	18	Y	NRSW	T	V	1	1	0	0	0	0
71	FTdx101D	NB on/off	navy	NB	19	Y	NBSW	T	V	1	1	0	0	0	0

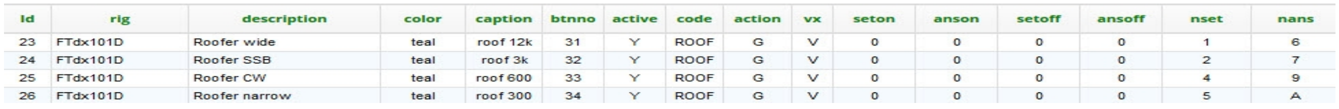
catcodes



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
5	FTdx101D	DNR on/off RxA	NRSW	A	4	5	5	NR0;	NR0u;	NR0u;
6	FTdx101D	DNR on/off RxB	NRSW	B	4	5	5	NR1;	NR1u;	NR1u;

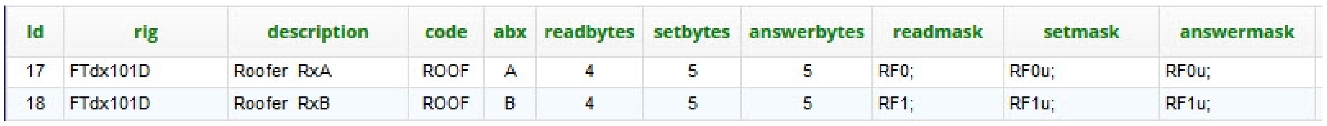
Roofing filter: select *This has four records in the buttons table and two records in catcodes*
The are four buttons and so four entries in the buttons table, all with **action = G** (grouped)
Each button has **code** = ROOF and **cset** = the roofer selector (FTdx101D CAT manual)
Example URL client to server:: &code=ROOF&jobdata=2 where 2 is the **cset** value

FTdx101D configuration shown below
buttons



Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset	nans
23	FTdx101D	Roofer wide	teal	roof 12k	31	Y	ROOF	G	V	0	0	0	0	1	6
24	FTdx101D	Roofer SSB	teal	roof 3k	32	Y	ROOF	G	V	0	0	0	0	2	7
25	FTdx101D	Roofer CW	teal	roof 600	33	Y	ROOF	G	V	0	0	0	0	4	9
26	FTdx101D	Roofer narrow	teal	roof 300	34	Y	ROOF	G	V	0	0	0	0	5	A

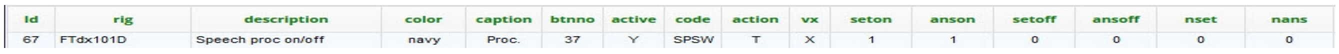
catcodes



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
17	FTdx101D	Roofer RxA	ROOF	A	4	5	5	RF0;	RF0u;	RF0u;
18	FTdx101D	Roofer RxB	ROOF	B	4	5	5	RF1;	RF1u;	RF1u;

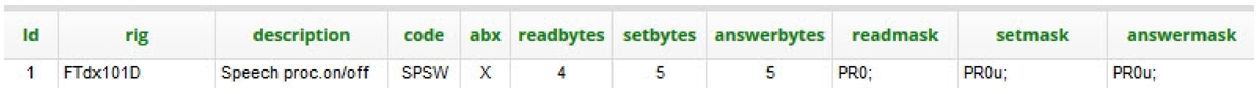
Speech proc on/off ***This has one record in the buttons table and one record in catcodes.***
code = SPSW **abx = X** **vx = X** **action = T** (toggled)

FTdx101D configuration shown below
buttons



Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset	nans
67	FTdx101D	Speech proc on/off	navy	Proc.	37	Y	SPSW	T	X	1	1	0	0	0	0

catcodes



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
1	FTdx101D	Speech proc.on/off	SPSW	X	4	5	5	PR0;	PR0u;	PR0u;

4.4 piWebCAT - Database table - sliders

See sliders configuration notes   timing.disable

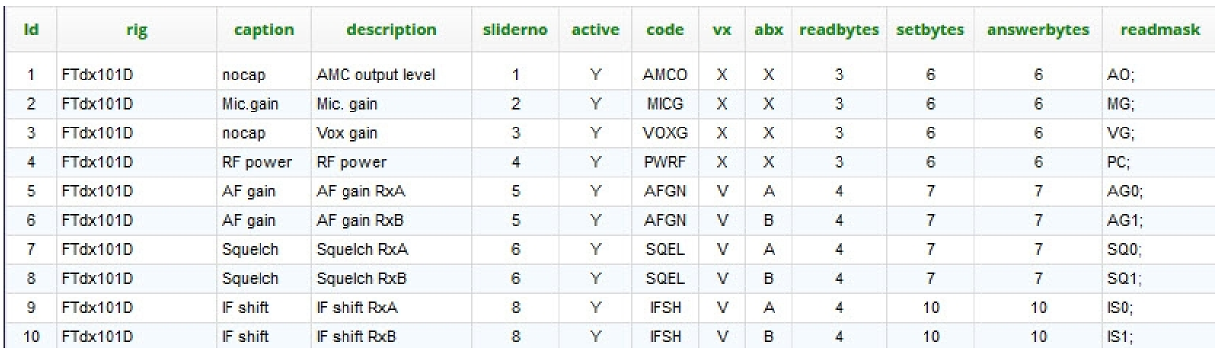
The configuration of **sliders** follows the same principles as for buttons.
The difference is that all *the client and server data is in one large table*
(where as **buttons** referenced table **catcodes** for server functions)
So - at startup, piWebCAT extracts part of the data from **sliders** to store on the client.
The server, on receipt of a command, reads **sliders** from the database for server functions.

The client held data from the **sliders** table includes data for formatting of the numeric text displays.
Slider positions are set when read from the radio at startup and change on slider movement.

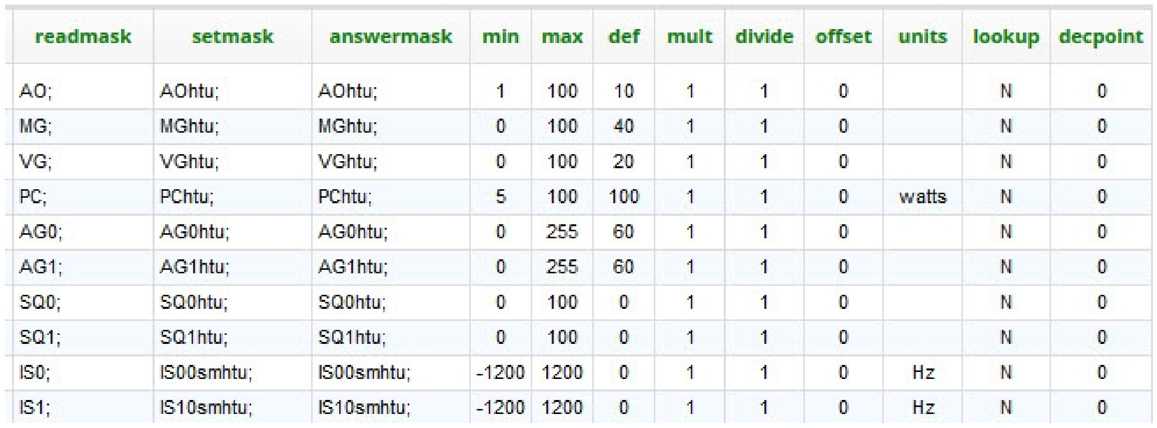
slider fields:

- **rig** The current radio - drop down selector (from **radios** table)
  Must have same spelling through the tables.
- **description** Descriptive text- no function
- **caption** The caption to the left of the slider.
  Note that this only applies to the central column of sliders with no adjacent on/off button with identifying caption. (suggest set to 'nocap' if not used)
- **sliderno** The sliders's unique, fixed, numeric identifier. See: Button and slider numbering
- **active** Y = slider active, N = slider inactive, S = sync (repetetive updates from rig)
  L = sync with alternative appearance,
- **code** 3 or 4 upper case characters. This links a slider movement action on the client to its action processes on the server. It is transmitted to the server with data (jobdata)
  See below for more explanation.
- **vx** **vx = V, X or U**. Client field that controls storage of latest setting for each VFO.
  vx is set to V for dual receiver settings for the radios stores a different value per receiver.
- **abx** **A** or **B** if there is pair of entries one for each VFO eg: DNR level, RF gain
  X if A or B not relevant. (eg: RF power level, mic gain etc) See: vx and abx
- **readbytes** The number of bytes in a read command (= no of chars in readmask)
- **setbytes** The number of bytes in a set command (= no of chars in setmask)
- **answerbytes** The number of bytes in an answer (= no of chars in answermask)
- **readmask** The character pattern of the read command, see Command masks
- **setmask** The character pattern of the set command. see Command masks
- **answermask** The character pattern of the answer. see Command masks
- **min** The minumum of the CAT data Ie before scaling etc)
- **max** **The maximum value of the CAT data.**
  min and **max** ensure that the range of the sent CAT value spans the limits of the slider and that the slider response to radio initiated change also fits the range,
- **def** Default value - set by the associated reset button (not available on all sliders)
- **mult** Multiplier to scale CAT value for display.
- **divide** Divider to scale CAT value for display.
  eg: use (CAT * 100) / 255 to scale 0-255 to 0 - 100
  (Multiply first then divide because we are working in integers).
- **offset** Offset applied to display value. eg: offset = -50 scales 0 -100 to -50 to +50.
- **units** Units after displayed value eg: Hz kHz etc
- **lookup** If lookup = **Y** then displayed value is taken from lookup table entries with matching code.
  Lookup uses the CAT value after scaling (if any).
  If lookup = **M** then displayvalue from lookup table entry with matching **code** and lookup table **mode** field = current operating mode (eg: USB, LSB etc).
  For lookup = **M**, lookup table mode must be the mode button's caption -
  ie: LSB, USB, CW etc (NOT SSB) See Lookup table .
- **decpoint** Add decimal point = dec places. eg: **decpoint** = 3 for 3000Hz to 3.000kHz.

Below is a screen dump of the slider table in piWebCAT editor
It is split into left and right parts with **readmask** in both for visual alignment.

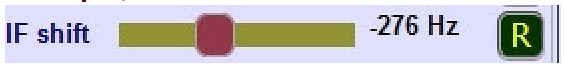


Id	rig	caption	description	sliderno	active	code	vx	abx	readbytes	setbytes	answerbytes	readmask
1	FTdx101D	nocap	AMC output level	1	Y	AMCO	X	X	3	6	6	AO;
2	FTdx101D	Mic.gain	Mic. gain	2	Y	MICG	X	X	3	6	6	MG;
3	FTdx101D	nocap	Vox gain	3	Y	VOXG	X	X	3	6	6	VG;
4	FTdx101D	RF power	RF power	4	Y	PWRF	X	X	3	6	6	PC;
5	FTdx101D	AF gain	AF gain RxA	5	Y	AFGN	V	A	4	7	7	AG0;
6	FTdx101D	AF gain	AF gain RxB	5	Y	AFGN	V	B	4	7	7	AG1;
7	FTdx101D	Squelch	Squelch RxA	6	Y	SQEL	V	A	4	7	7	SQ0;
8	FTdx101D	Squelch	Squelch RxB	6	Y	SQEL	V	B	4	7	7	SQ1;
9	FTdx101D	IF shift	IF shift RxA	8	Y	IFSH	V	A	4	10	10	IS0;
10	FTdx101D	IF shift	IF shift RxB	8	Y	IFSH	V	B	4	10	10	IS1;

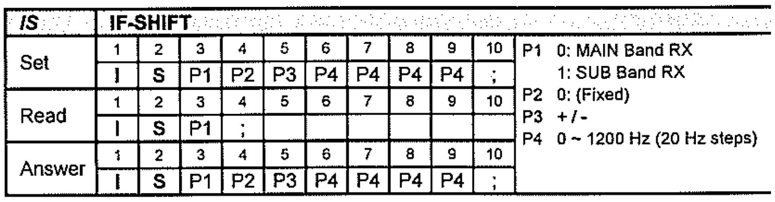


readmask	setmask	answermask	min	max	def	mult	divide	offset	units	lookup	decpoint
AO;	AOhtu;	AOhtu;	1	100	10	1	1	0		N	0
MG;	MGhtu;	MGhtu;	0	100	40	1	1	0		N	0
VG;	VGhtu;	VGhtu;	0	100	20	1	1	0		N	0
PC;	PChtu;	PChtu;	5	100	100	1	1	0	watts	N	0
AG0;	AG0htu;	AG0htu;	0	255	60	1	1	0		N	0
AG1;	AG1htu;	AG1htu;	0	255	60	1	1	0		N	0
SQ0;	SQ0htu;	SQ0htu;	0	100	0	1	1	0		N	0
SQ1;	SQ1htu;	SQ1htu;	0	100	0	1	1	0		N	0
IS0;	IS00smhtu;	IS00smhtu;	-1200	1200	0	1	1	0	Hz	N	0
IS1;	IS10smhtu;	IS10smhtu;	-1200	1200	0	1	1	0	Hz	N	0

Example, IF shift



Yaesu CAT manual



IS	IF-SHIFT										
Set	1	2	3	4	5	6	7	8	9	10	P1 0: MAIN Band RX
	I	S	P1	P2	P3	P4	P4	P4	P4	;	1: SUB Band RX
Read	1	2	3	4	5	6	7	8	9	10	P2 0: (Fixed)
	I	S	P1	;							P3 + / -
Answer	1	2	3	4	5	6	7	8	9	10	P4 0 ~ 1200 Hz (20 Hz steps)
	I	S	P1	P2	P3	P4	P4	P4	P4	;	

This is a slider without an on/off button but with a reset-to default button.
caption = "IF shift" for the label
code = "IFSH" (or whatever you choose) It is loaded to the client at startup and transmitted
to the server as a linking identifier.
sliderno is fixed for this slider, irrespective of its function and must not be changed.
abx = A or B.Current VFO selection A or B is held on the client and sent to the server.
The server can then send the correct (A or B) CAT message to the radio.
readbytes or **setbytes** and **readmask** or **setmask** define the message to the radio.
answrerbytes is essential so that the server code knows how many returned bytes to wait for.
answermask is is used to decode the character string response for the radio.
min. max, multi, divide, offset, units and **lookup** define the presentation of the data, ie: **-276Hz**
The CAT manual specifies the data in P2 P3 P3 P3 P3 eg: **- 0276**
P2 is the sign (- or +). This is represented by lower case 's; in the mask.

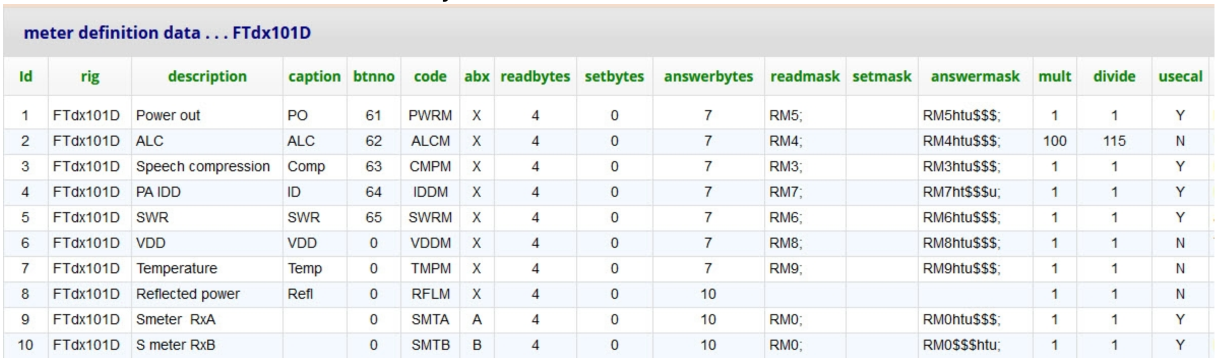
4.5 piWebCAT - Database table - meter

piWebCAT has an S meter on receive and five button-selected meter options on transmit.

The five Tx buttons are set up in the **buttons** table.
The **meter** table controls the subsequent repetitive meter reading.
The **timing** table sets the meter repetition interval

The link between the **buttons** table and the **meter** table is the **btnno** field values, 61 to 65.

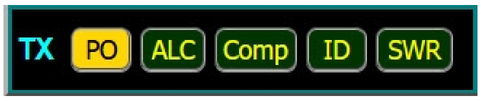
The meter table is shown below for my FTdx101D



meter definition data . . . FTdx101D

Id	rig	description	caption	btnno	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask	mult	divide	usecal
1	FTdx101D	Power out	PO	61	PWRM	X	4	0	7	RM5;		RM5htu$$$;	1	1	Y
2	FTdx101D	ALC	ALC	62	ALCM	X	4	0	7	RM4;		RM4htu$$$;	100	115	N
3	FTdx101D	Speech compression	Comp	63	CMPM	X	4	0	7	RM3;		RM3htu$$$;	1	1	Y
4	FTdx101D	PA IDD	ID	64	IDDM	X	4	0	7	RM7;		RM7ht$$$u;	1	1	Y
5	FTdx101D	SWR	SWR	65	SWRM	X	4	0	7	RM6;		RM6htu$$$;	1	1	Y
6	FTdx101D	VDD	VDD	0	VDDM	X	4	0	7	RM8;		RM8htu$$$;	1	1	N
7	FTdx101D	Temperature	Temp	0	TMPM	X	4	0	7	RM9;		RM9htu$$$;	1	1	N
8	FTdx101D	Reflected power	Refl	0	RFLM	X	4	0	10				1	1	N
9	FTdx101D	Smeter RxA		0	SMTA	A	4	0	10	RM0;		RM0htu$$$;	1	1	Y
10	FTdx101D	S meter RxB		0	SMTB	B	4	0	10	RM0;		RM0$$$htu;	1	1	Y

The are two S meter records (RxA and RXb) and eight Tx meter records (reflect power not available)
The S meter **code** fields SMTA and SMTB are fixed on the client and so must be entered as these so that the server will recognise them.
The Tx meter **code** fields are copied to the client at startup and so you can use any **code** of your choice.
Five of the Tx meters have been assigned to the available five option buttons as shown below



In the meter table, these are the items with a non-zero **btnno** field (ie: 61 to 65)

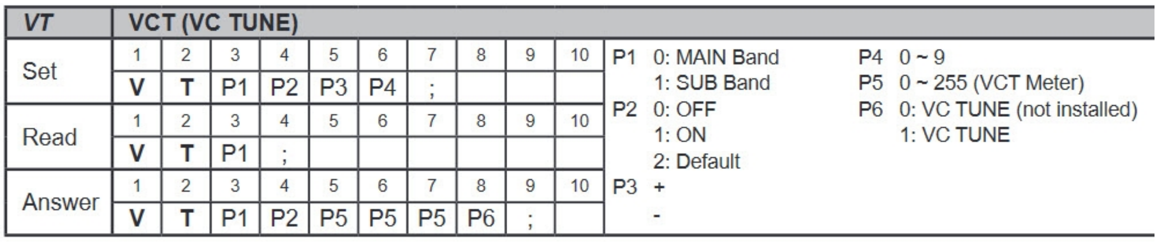
Meter table field list:

- **rig** The current radio - drop down selector (from radios table)
- **description** Descriptive text- no function
- **caption** The caption to the left of the slider. Note that this only applies to the central column of sliders with no adjacent on/off button with identifying caption.
- **btnno** The sliders's unique, fixed, numeric identifier as discussed earlier.
- **code** Specified once here. Used in message from client and recognised in server.
- **abx** If **A** or **B,** Client sends Vfo **A** or **B** to server. **abx** allows server to read appropriate meter. For Tx meters abx **= X** (not VFO-specific)
- **readbytes** The number of bytes in a read command (= no of chars in readmask)
- **setbytes** *Unused* - we do not set the meter! It exists for PHP code reasons.
- **answerbytes** The number of bytes in an answer (= no of chars in answermask)
- **readmask** The character pattern of the read command, see Command masks
- **setmask** *Unused* - we do not set the meter! It exists for PHP code reasons.
- **answermask** The character pattern of the answer. see Command masks
- **mult** Multiplier (default = 1)
- **divide** Divisor (default = 1) Meter CAT value is scaled by (CAT * mult) / divide ***before*** being optionally modified by the **metercal** table data.
- **usecal** Y or N. If Y, then the CAT value is processed by the calibration table. (20 point calibration for each meter with linear interpolation between points)

4.6 piWebCAT FTdx101D VCT example (VC tune - RF tuning)

The FTdx101D has VCT on the main receiver (A) but not on the sub receiver (B).
When switched in, it narrows the front end bandwidth.
(The spectrum scope display is therefore attenuated outside the VCT passband.)

Initially, I did not attempt set up VCT in piWebCAT because I found the information in the CAT manual rather unclear. This information is as follows:



VT	VCT (VC TUNE)									
Set	1	2	3	4	5	6	7	8	9	10
	V	T	P1	P2	P3	P4	;			
Read	1	2	3	4	5	6	7	8	9	10
	V	T	P1	;						
Answer	1	2	3	4	5	6	7	8	9	10
	V	T	P1	P2	P5	P5	P5	P6	;	

P1	0: MAIN Band
	1: SUB Band
P2	0: OFF
	1: ON
	2: Default
P3	+
	-
P4	0 ~ 9
P5	0 ~ 255 (VCT Meter)
P6	0: VC TUNE (not installed)
	1: VC TUNE

VCT on/off -
The fourth character in the set command, P2 clearly determines on or off.
I discovered by experimentation that P3 and P4 need to be 0 for this to work.

So the VCT ON and VCT OFF commands (VFO A) are: **VT0100;** and **VT0000;**
This is conveniently controlled by a toggling button (**action = T**)
and **catcodes.setmask** = **VT0u00;** (where **u** is the value sent from client **von** and **voff** fields.

VCT tuning
We have P3 marked as + or - and P4 with value 0-9.
I found that command VT01+6; gave an upwards tuning step and VT01-6; would step downward.

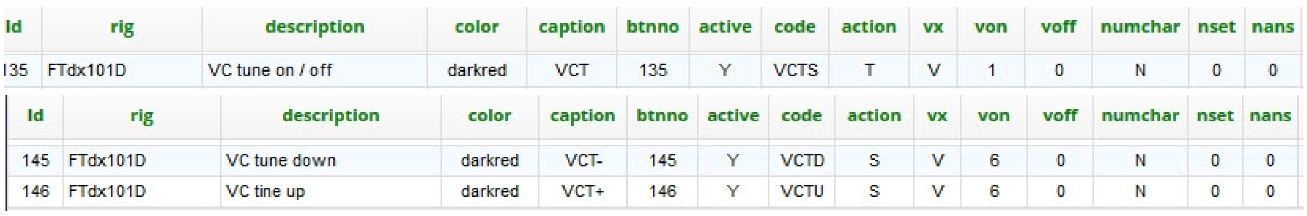
I choose a step setting of 6 from a stated available range of 0-9.
This traverses the tuning range on all bands in 28 steps - which gives an adequate resolution.

I configured two buttons:
VCT+ and VCT - together with the toggling VCT on/off button;

The **buttons** and **catcodes** table entries are shown below.
Note that there are two ways to specify the step of 6 units:

- A single action button (**action = S**) will send the **von** field = 6 as data to the server.
  I have set **von** = 6 and this is substituted for the u (units) in the **setmasks** = VT01+u; and VT01-u;
- An alternative would have been to ignore the von value and use setmasks = VT01+6; and VT01-6;)

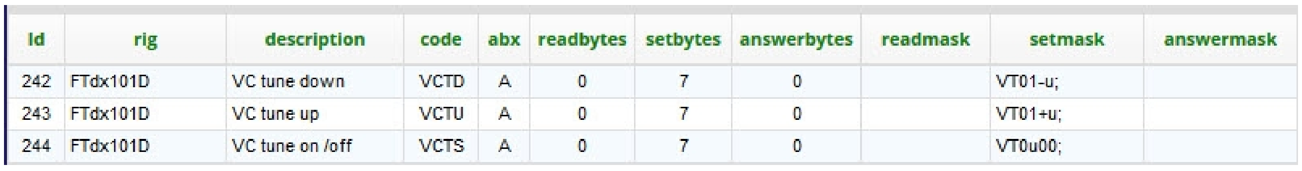
buttons table



Id	rig	description	color	caption	btnno	active	code	action	vx	von	voff	numchar	nset	nans
135	FTdx101D	VC tune on / off	darkred	VCT	135	Y	VCTS	T	V	1	0	N	0	0

Id	rig	description	color	caption	btnno	active	code	action	vx	von	voff	numchar	nset	nans
145	FTdx101D	VC tune down	darkred	VCT-	145	Y	VCTD	S	V	6	0	N	0	0
146	FTdx101D	VC tine up	darkred	VCT+	146	Y	VCTU	S	V	6	0	N	0	0

catcodes table



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
242	FTdx101D	VC tune down	VCTD	A	0	7	0		VT01-u;	
243	FTdx101D	VC tune up	VCTU	A	0	7	0		VT01+u;	
244	FTdx101D	VC tune on /off	VCTS	A	0	7	0		VT0u00;	

4.7 ASCII configuration - examples

Note that from the **buttons** table: Toggled buttons always send **von / voff.** Grouped always send **nset** or **chset**.

DNR on/off switching - single button

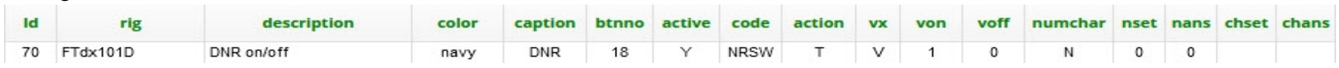
Single entry in buttons table action = T (toggling) **von** = 1 **voff** = 0

- on click, client transmits **code** = NRSW jobdata 1 or 0

Yaesu CAT manual: read: NR0; for VFO main NR1; for VFO sub
send and receive are NR0u; NR1u; where u is 1 - 9

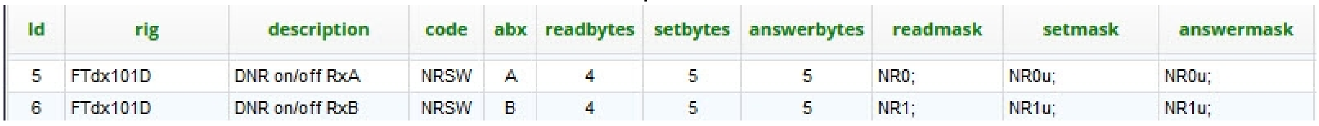
In the **buttons** record **vx = V** causes the client to send the current VFO (A or B) to the server.
At the server, the separate A and B records are identified by **abx** = A or B See: vx and abx

A single **buttons** record



Id	rig	description	color	caption	btnno	active	code	action	vx	von	voff	numchar	nset	nans	chset	chans
70	FTdx101D	DNR on/off	navy	DNR	18	Y	NRSW	T	V	1	0	N	0	0		

There are two **catcodes** records because there are separate commands for VFO A and VFO B



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
5	FTdx101D	DNR on/off RxA	NRSW	A	4	5	5	NR0;	NR0u;	NR0u;
6	FTdx101D	DNR on/off RxB	NRSW	B	4	5	5	NR1;	NR1u;	NR1u;

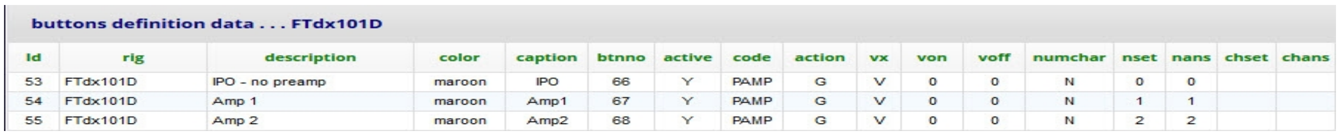
IPO / premp swiching - three buttons in a group of 3

There are three button records in **buttons** because we have chosen to have a group of three:
pre-amp-off (IPO), Amp1 and Amp2.
action = G (grouped) **code** = PAMP **nset** and **nans** = **0** (IPO), **1** (Amp1) and **2** (Amp2)

There are separate **nset** and **nans** values because a small number of commands have different set and read values.
Likewise, at least one command uses character rather than numerics, hence the alternative **chset** and **chans** fields.
The **numchar** field determines whether num or char (N or C)

The FTdx101D CAT manual shows read commands: PA0; (main) and PA1; (sub)
The corresponding set and answer formats are: PA0u; and PA1u; where u is 0, 1 or 2.

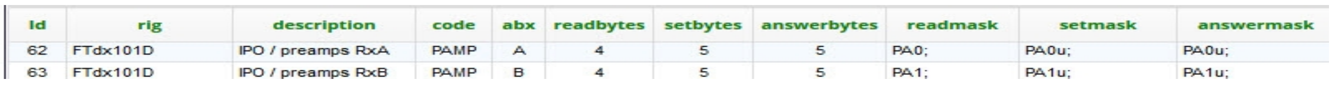
buttons records



buttons definition data . . . FTdx101D

Id	rig	description	color	caption	btnno	active	code	action	vx	von	voff	numchar	nset	nans	chset	chans
53	FTdx101D	IPO - no preamp	maroon	IPO	66	Y	PAMP	G	V	0	0	N	0	0		
54	FTdx101D	Amp 1	maroon	Amp1	67	Y	PAMP	G	V	0	0	N	1	1		
55	FTdx101D	Amp 2	maroon	Amp2	68	Y	PAMP	G	V	0	0	N	2	2		

catcodes records



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
62	FTdx101D	IPO / preamps RxA	PAMP	A	4	5	5	PA0;	PA0u;	PA0u;
63	FTdx101D	IPO / preamps RxB	PAMP	B	4	5	5	PA1;	PA1u;	PA1u;

Note that we have two catcodes records, one for the each VFO because the FTdx101D stores a different status
for each VFO. The **abx =A** and **abx = B** records contain the command masks for the A and B VFOs.

RF power level - slider

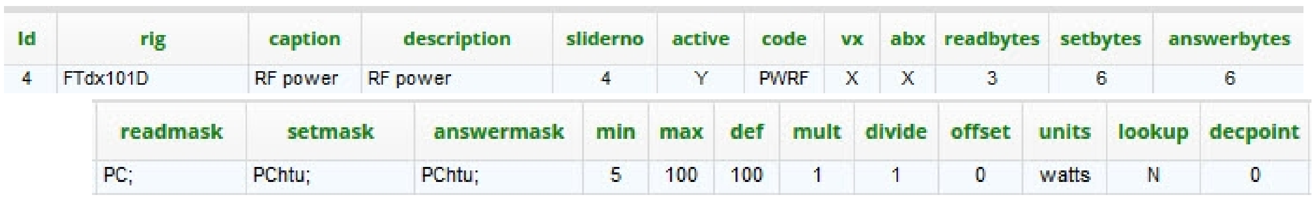
*Slider data for client **and** server functions all in the **slidersciv** table.*

For **sliders**, there are no data fields in the table. (The data is derived from slider position)
The table specifies **min** and **max**. These are always CAT value min and max (not displayed min and max)
When you move the slider, the value sent to the radio is based on **min** at slider left and **max** at slider right.
So here, min = 5 max = 100 (So, for example, at mid position the data to the radio is 52.)

FTdx101D CAT manual has: read PC; set PChtu; answer PChtu;
where htu = data (hundreds, tens and units)

This is not a VFO dependent setting, so there is only one record in the table with vx **= X** and **abx = X**.
vx = X means that the command is sent from the client with parameter rxab = X See: vx and abx
(whereas if vx = V, the command from the client is sent as rxab = A or B according to current VFO selection)

sliders table



Id	rig	caption	description	sliderno	active	code	vx	abx	readbytes	setbytes	answerbytes
4	FTdx101D	RF power	RF power	4	Y	PWRF	X	X	3	6	6

readmask	setmask	answermask	min	max	def	mult	divide	offset	units	lookup	decpoint
PC;	PChtu;	PChtu;	5	100	100	1	1	0	watts	N	0

Note: RF power is the only setting that is *always* loaded from the radios on VFO change (rather than use stored)
*For this to happen, you must use **code = PWRF.***

Read and write Frequency

Not a slider nor a button but driven from the piWebCAT tuners.
Configured in the **catcodes** table

Frequency setting:
Mouse and thumbwheel actions modifiy variable **freqSet** in the client.
A timer driven task monitors **freqSet** and on change queues a message to the server.
This client activity is hard coded - there is no database configuration for it.

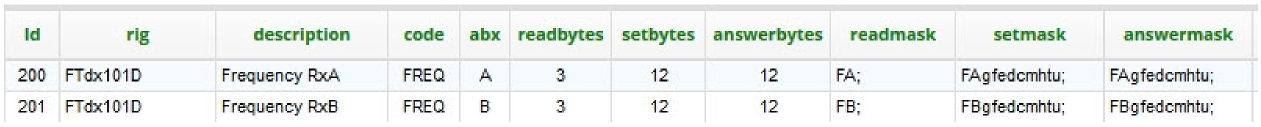
The control codes for the radios are configured in the **catcodes** table.
They use **code = FREQ** and **abx = A** for VFOA and **code = FREQ** and **abx = B** for VFOB.
FREQ is hard coded in the client and so must not be changed.

From the FTdx101D CAT manual:
- read main VFO freq. is FA; set and answer commands are, for example FA028123456;
- read sub VFO freq. is FB; set and answer commands are, for example FB028123456;

ie the data is a nine digit frequency. The mask for this is: FAgfedcmhtu; etc
where u = units, t = tens, h = hundreds, m = thousands, c = ten-thousands etc

abx = A or B to identify the record in order to match the command to the VFO selection from the client.
The client sends a command as if a slider command ... but it is internally generated as above.

catcodes table



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
200	FTdx101D	Frequency RxA	FREQ	A	3	12	12	FA;	FAgfedcmhtu;	FAgfedcmhtu;
201	FTdx101D	Frequency RxB	FREQ	B	3	12	12	FB;	FBgfedcmhtu;	FBgfedcmhtu;

Band change buttons etc

The following is repeated in How it works.

The FTdx101D CAT manual documents the **band select** command: BStu;   where tu = band code 00 to 10.
This is a write-only command. There is no command to read the current band from the radio. We therefore have the following arrangement: (actual timing intervals specified in **timings** table per radio)

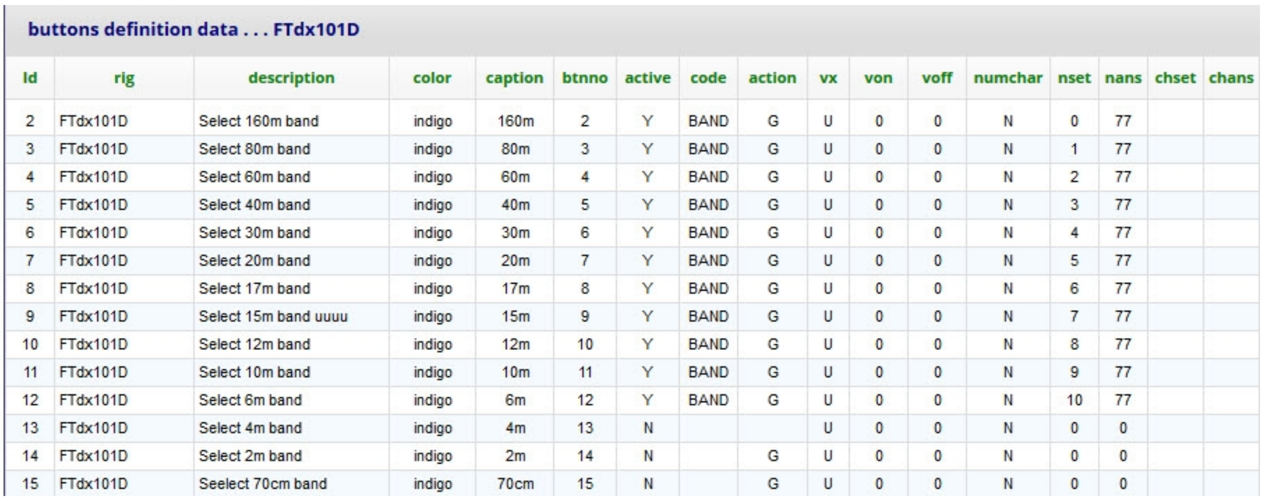
Automated repetitive tasks to;

- Read main (A) frequency from radio every 200ms.
- Read sub (B) frequency from radio every 600ms.
- Check the latest read of main (A) frequency for band status every 1000ms.
  (piWebCAT has hard coded band limits.)
  If in a new band, then draw tuning scale for new band and highlight appropriate band button.
  If frequency goes out of band, change frequency marker from red to olive. Leave it at band edge.
  Do not redraw band scale until we enter a new band.

Note that the band limits are in database table **bands**.
However, the spans of the fourteen band tuning scales are hard coded.

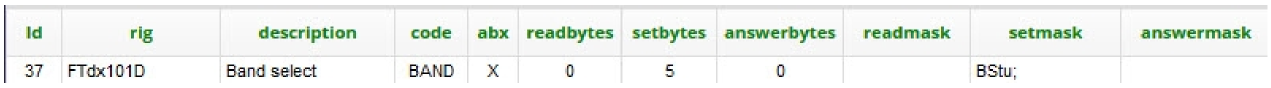
Band buttons

The band button sends a band change command.
The radio will change to a frequency in the new band.
This will be the last used frequency in that band (or according to radios normal band change behaviour)
piWebCAT will pick up the resulting frequency change by its ongoing repetitive frequency read process)



buttons definition data . . . FTdx101D

Id	rig	description	color	caption	btnno	active	code	action	vx	von	voff	numchar	nset	nans	chset	chans
2	FTdx101D	Select 160m band	indigo	160m	2	Y	BAND	G	U	0	0	N	0	77		
3	FTdx101D	Select 80m band	indigo	80m	3	Y	BAND	G	U	0	0	N	1	77		
4	FTdx101D	Select 60m band	indigo	60m	4	Y	BAND	G	U	0	0	N	2	77		
5	FTdx101D	Select 40m band	indigo	40m	5	Y	BAND	G	U	0	0	N	3	77		
6	FTdx101D	Select 30m band	indigo	30m	6	Y	BAND	G	U	0	0	N	4	77		
7	FTdx101D	Select 20m band	indigo	20m	7	Y	BAND	G	U	0	0	N	5	77		
8	FTdx101D	Select 17m band	indigo	17m	8	Y	BAND	G	U	0	0	N	6	77		
9	FTdx101D	Select 15m band uuuu	indigo	15m	9	Y	BAND	G	U	0	0	N	7	77		
10	FTdx101D	Select 12m band	indigo	12m	10	Y	BAND	G	U	0	0	N	8	77		
11	FTdx101D	Select 10m band	indigo	10m	11	Y	BAND	G	U	0	0	N	9	77		
12	FTdx101D	Select 6m band	indigo	6m	12	Y	BAND	G	U	0	0	N	10	77		
13	FTdx101D	Select 4m band	indigo	4m	13	N			U	0	0	N	0	0		
14	FTdx101D	Select 2m band	indigo	2m	14	N		G	U	0	0	N	0	0		
15	FTdx101D	Seelect 70cm band	indigo	70cm	15	N		G	U	0	0	N	0	0		



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
37	FTdx101D	Band select	BAND	X	0	5	0		BStu;	

All band buttons send from client to server with **code** = **BAND**   **abx** = **X**   and **nset** = band code (0 - 10)
abx = **X** because the command is not VFO specific. Its action on the radio is on the current VFO

Note: If other RS232 radios have two band commands, one for each VFO, then we would have to configure piWebCAT to apply the band change to the current VFO.
This would be done as follows:
  Set vx = V in every band-button **buttons** table record.
  This will make the client send A or B according to current VFO selection.
  Create two **catcodes** records, one with CAT commands for VFO A and abx = A
  and one with CAT commands for VFO B and with abx = B.

Memory selection buttons - examples

FTdx101D has a MW (memory write) command. This is write-only to set up a memory channel.
The command is 28 characters in length and sets a list of parameters to a channel.
piWebCAT does not support this. Memories must be set up on the radio.
Pre-configured memories can then be ***selected*** by piWebCAT.

The **MC** command selects a memory channel and applies it to the current VFO. I use this.

There are MA and MB commands to apply the selected memory to VFO A or B.
I do not understand what purpose these have. There is no command to select a memory channel without applying it to the VFO, so I see no use in piWebCAT for MA and MB.

piWebCAT has two fixed values for buttons and catcodes code fields:

- **MECH** should be used for a single memory channel button (eg: for calling channel etc)
- **MPAD** causes the button to automatically launch a numeric keypad memory selector.
  The OK button on the keypad then sends an **MPAD** command to the server with the selected memory as numeric data.

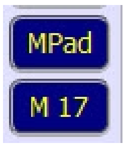
Both **MECH** and **MPAD** are followed after one second by a **code = MTOV** command to the server.
For the FTdx101D, we do not configure this MTOV command in **catcodes** and so it is simply lost.

If you have a radio whose 'MC' command doesn't apply the selected memory to a VFO, then you need to configured a **code = MTOV** in **catcodes** to send a memory-to-VFO command to the radio.
This is done for Icom radios - see Icom examples.

FTdx101D CAT manual: memory channel selector command is:
read: **MC;** set **MChtu**; answer **MChtu;** where htu is the channel number, eg: 017

This command selects the memory and applies it to the VFO.

I configured 2 spare buttons in my FTdx101D to manage the radio's memories.



MPad displays a numeric keypad for memory channel selection.
The keypad OK button selects a channel.

M 17 selects memory channel 17.

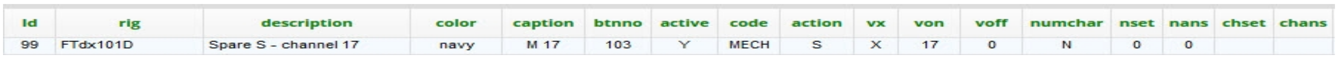
Each button must be configured in the **buttons** table (client settings)
and in the **catcodes** table (server / CAT settings)
*You can have multiple single channel (MECH) buttons in the **buttons** table sharing a single **catcodes** record.*

M 17 - single channel buttons *(- no command-specific coding)*

The **buttons** records control what happens on the client.
The **catcodes** record controls what happens on the server ... ie: sending to the radio.
buttons.action = 'S'. These are single click buttons (ie not grouped with others, not toggling on/off)
They are not VFO specific so I set **buttons.vx = 'X'** **catcodes.abx = 'X'**
The data field with **button.action = 'S'** is **von** -- which is set to 17.
(We must use **code = 'MECH'** if a following copy memory to VFO CAT command is needed.)

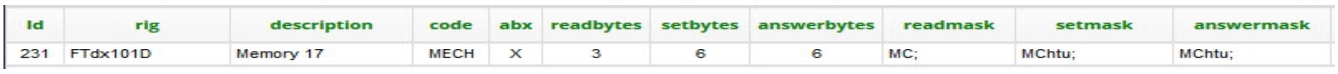
The database records for the M17 memory select button are shown below.

buttons



Id	rig	description	color	caption	btnno	active	code	action	vx	von	voff	numchar	nset	nans	chset	chans
99	FTdx101D	Spare S - channel 17	navy	M 17	103	Y	MECH	S	X	17	0	N	0	0		

catcodes



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
231	FTdx101D	Memory 17	MECH	X	3	6	6	MC;	MChtu;	MChtu;

Memory selector

The button is programmed to display a popup numeric key pad as a channel selector.
The code field must be set to **MPAD**, both in **buttons** (to display the popup) and **catcodes** (to respond correctly)
Note that this is a special, hardcoded two-stage command system.
The button displays the popup.
The selected channel is sent client > server > radio when the OK button on the keypad is clicked.

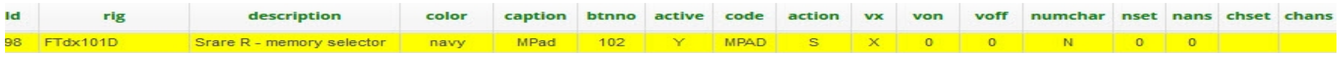
One second later, a **code = MTOV** is automatically sent (no **buttons** configuration needed for this)
MTOV is intended for a command to copy selected memory to VFO.
The FTdx101D doesn't need it - so we simply do not configure it in **catcodes**.

buttons: **action = 'S' code = 'MPAD'** data fields **von**, **voff etc** are all unused.
(The data is the keypad selection)

The CAT command is exactly the same as **'M 17'** above,
ie: it sets a numeric channel (but from the keypad rather than fixed).

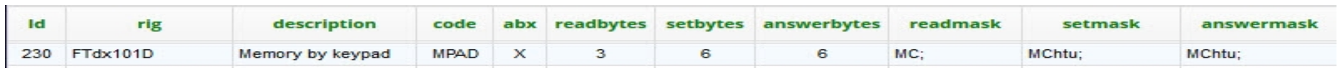
The database records for these memory select buttons are shown below.

buttons



Id	rig	description	color	caption	btnno	active	code	action	vx	von	voff	numchar	nset	nans	chset	chans
98	FTdx101D	Srare R - memory selector	navy	MPad	102	Y	MPAD	S	X	0	0	N	0	0		

catcodes



Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
230	FTdx101D	Memory by keypad	MPAD	X	3	6	6	MC;	MChtu;	MChtu;

Monitor on/off switching and level

This is included as an example of a single CAT command: **ML...** being used for switching and level.

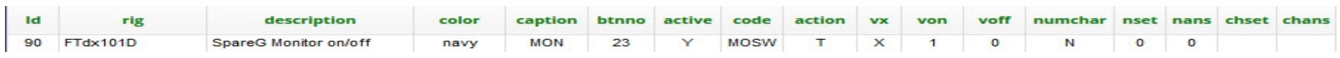
The CAT manual shows:

- **M L P1 P2 P2 P2 ;** where **P1** is a single digit number and **P2** is a three digit number.
- If **P1** = 0 then **P2** is on/off with P2 = 0 for off and P2 = 1 for on.
- If **P1** = 1 then **P2** is level 0 - 100.

So for on/off we use:
readbytes = 4, setbytes = 7, answerbytes = 7, readmask = ML0; setmask = ML0htu, answermask = ML0htu;

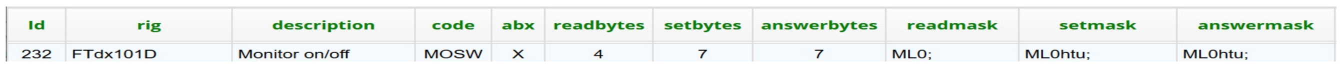
and for level we use:
readbytes = 4, setbytes = 7, answerbytes = 7, readmask = ML1; setmask = ML1htu, answermask = ML1htu;

buttons table - montior on/off



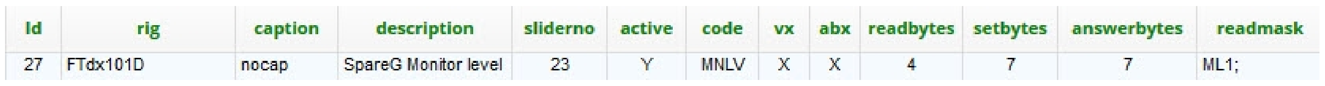
Id	rig	description	color	caption	btnno	active	code	action	vx	von	voff	numchar	nset	nans	chset	chans
90	FTdx101D	SpareG Monitor on/off	navy	MON	23	Y	MOSW	T	X	1	0	N	0	0		

catcodes table - monitor on/off

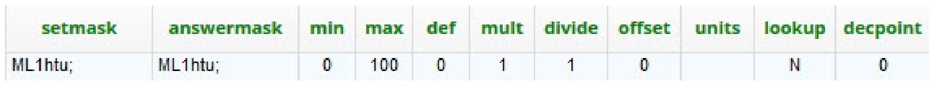


Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
232	FTdx101D	Monitor on/off	MOSW	X	4	7	7	ML0;	ML0htu;	ML0htu;

sliders table - monitor level



Id	rig	caption	description	sliderno	active	code	vx	abx	readbytes	setbytes	answerbytes	readmask
27	FTdx101D	nocap	SpareG Monitor level	23	Y	MNLV	X	X	4	7	7	ML1;



setmask	answermask	min	max	def	mult	divide	offset	units	lookup	decpoint
ML1htu;	ML1htu;	0	100	0	1	1	0		N	0

5.1 Yaesu older 5-byte command radios

Modern Yaesu radios transmit and receive CAT data in the form of readable ascii text.
eg: **FA145500000;**

Kenwood and Elekraft use a similar system (Many Kenwood codes are the same as Yaesu)

In piWebCAT the configuration for these radios is selected in the **rigs** table as **ASCII**.
The ASCII configuration system is described in detail earlier in this document.
It uses masks where selected lower case letters map to the data within the command.
All other mask characters are included in the command verbatim.

Earlier Yaesu radios used CAT data constructed of hexadecimal and binary coded decimal bytes.
All commands to the radio are 5 bytes. Returned data packets from the radio can be from 1 to 32 bytes.
These radios include: FT847, FT817, FT818, FT920, FT890, FT1000, FT1000MP mkv.

Unlike the modern text configured radios, they do not have matching pairs of read and write commands.
Setting and reading commands are completely separate and differently structured.
For example:

- **FTdx101D** - RF power: Set 95w is PL095; Read is PL; with answer PL095;
- **FT920** - A relatively limited range of setting commands - no RF power setting.
  Received data: Status updates: eg: 28 byte VFO A and B freq, mode, clarifier block.
  Status flags - returns 8 bytes with a variety of bitwise information.
  Read meter: returns S-meter. power etc
  - but also NR, PROC and squelch levels (NO CAT setting of these)

I have had an **FT847** for many years.
This has a very limited set of CAT commands ... including switching of the unique satellite features.
I used this to develop a configuration system for the 'five byte' system.

The **FT818** is similar but with a few extra commands.
I had some help in testing from a FT818 owner during development and so my supplied database contains an FT818 configuration.

I then acquired an **FT920** and developed a configuration database for it.
The FT920 has a much larger set of commands and so my supplied FT920 database provides a good starting point for a user to configure for FT1000 etc.

The YAESU5 configuration option
The configuration is selected in the rigs table as: **YAESU5**.
*It uses the same tables as the **ASCII** configurations.*
The user enters commands in fields: **readmask**, **setmask** and **answermask** as text characters.
The outgoing read and set commands to the radio are configured as ten characters in **readmask** and **setmask.**
The RPi web server translates the ten characters to five bytes for transmission to the radio.

answermask is dealt with differently. (otherwise, a mask of 56 characters would be required to decode 28 bytes)

answermask configuration text is of the form #08:03:01:1Fbb
= receive 08 bytes. Extract 01 byte starting at byte 3. Perform a logical AND with hex 1F to return bits 0 - 4.
(This is discussed in detail and with examples later)

The system allows the use of easily interpreted text to configure a somewhat more complex CAT system.

The five byte outgoing commands are the easiest to deal with:
eg: FT847 read frequency and mode is: specified as five bytes: * * * * E7 where * are don't care bytes.
So I entered readmask = 00000000E7 as text
The server converts this to hexadecimal bytes: 00 00 00 00 0xE7

5.2 YAESU5 configuration option

FT847, FT817, FT818

These radios have a quite limited CAT control command set. They include the following features:

- All outgoing commands are 4 bytes optional BCD data followed by a single byte hexadecimal opcode.
- There are three 'read' commands:
  - **Frequency + mode** returning five bytes. Bytes 1 to 4 (BCD) = freq / 10. Byte 5 = mode
  - **Receive status** returning a single binary byte. Bits 0 to 4 = S meter (0-31)
    Bit 5 - discriminator / mode ?? Bit 6 CTCSS status. Bit 7 squelch status.
  - **Transmit status** returning a single binary byte. Bits 0 - 4 = Tx power. Bit 5 = split on/off.
    Bit 6 = high SWR. Bit 7 = PTT (0 = Tx, 1 = Rx)
- The command set is quite limited compared to the FT920, FT1000 and more modern radios.
  This leaves piWebCAT's screen looking relatively unpopulated.
  The available commands differ between radios.
  eg: the FT847 CAT does not have control of split, VFO A/B toggle or clarifier.

FT847: Early FT847s had no status read capability. A firmware update for this is not now available.
Fortunately, my FT847 does have read capability. I could therefore test S meter, power meter etc.
Its serial number suggests that it would have needed the firmware update.
I have no recollection of the upgrade . But I have had the radio for ? 17 years !!!

The initial development was done on my FT847 with support from an FT818 user.
I used the FT818 manual (with comparisons to the FT847 manual).

The CAT manuals do not actually state that Rx and Tx status requests return only one byte.
This caused me confusion until I found the following very useful internet reference:

http://www.ka7oei.com/ft817_meow.html

Hardware - connection:

- The FT847 has a 9 pin D RS232 connector which requires a crossover (null modem) cable.
- The FT817 and FT818 needs a Yaesu CT-62 cable ... or home made equivalent.

5.3 FT920

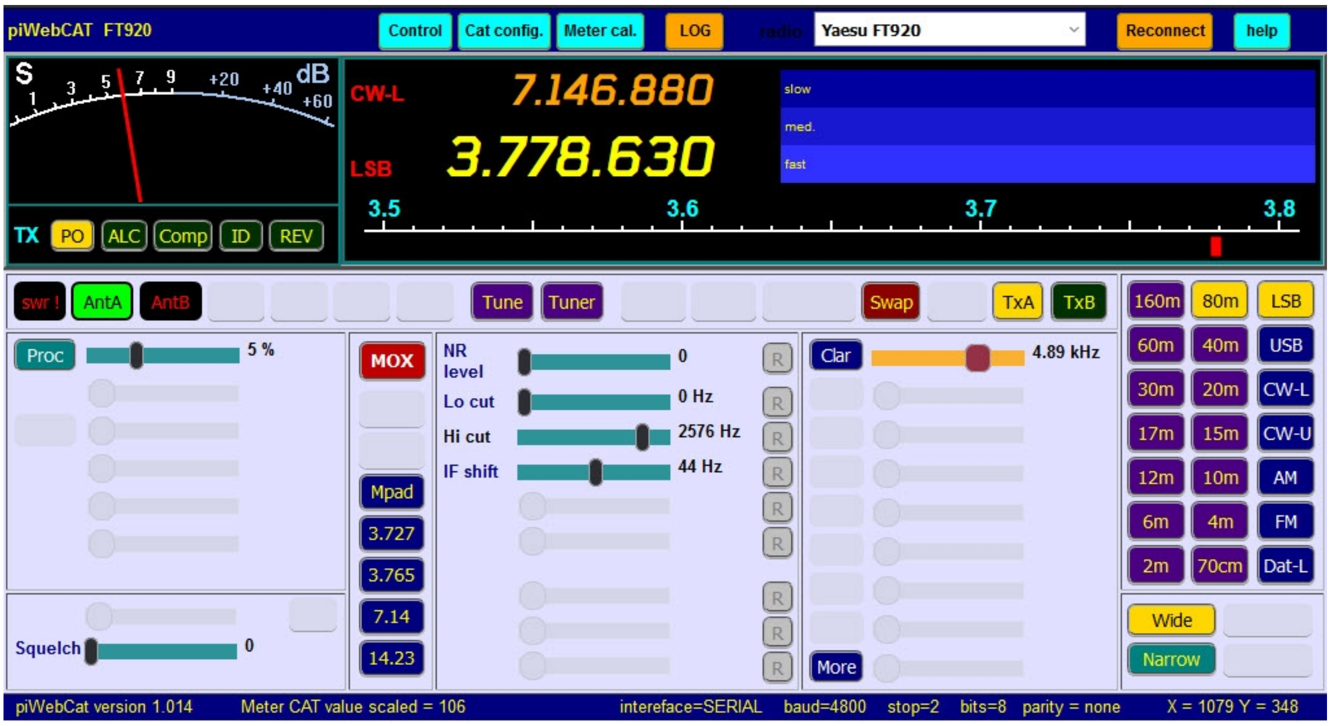
These radios use the Yaesu 5 byte command system with a much more extensive command set compared to the FT817 etc.
Their CAT capabilities are however still much less than the modern radios where most commands have matching read and write facilities.
For example: on the FTdx101D, I assigned useful functions to all 90 buttons and 27 sliders.
On the FT920, I could not achieve this.

My current FT920 control window is shown below.

Note the sliders:

- Clarifier is fully read/write with slider **active = S** - so that it both controls the radio and follows changes on the radio.
- The other six sliders have **active = '**L' which periodically updates a small black indicator on a teal coloured slider.
  These are read only because FT920 CAT only supports reading of these items (not setting)

Note the SWR! button on the left. This again is ready only. It has active - L (LED)
It lights up in a bright colour of my choice to give an high SWR warning from the radio.



CW-L 7.146.880
LSB 3.778.630

Proc 5 %
Squelch 0
NR level 0
Lo cut 0 Hz
Hi cut 2576 Hz
IF shift 44 Hz
Clar 4.89 kHz

```
piWebCat version 1.014      Meter CAT value scaled = 106          intereface=SERIAL   baud=4800   stop=2   bits=8   parity = none      X = 1079 Y = 348
```

I purchased an FT920, partly to be able to support this group of radios with piWebCAT.
My user guide downloaded from Yaesu is dated 1997. Its CAT section appears to have some errors.

My first problem was that CAT did not work at all. The RS232 Tx output was +1v (It should be -10v)
All was subsequently well after tricky micro surgery to replace the MAX232CWE RS232 driver chip.

The supplied SD card image now includes a working FT920 configuration in the database.
The FT920 development resulted in:

- A option for having regular updates to the state of selected read-only button and slider controls.
- A rigfix option for specific areas of translation from ascii text configuration to and from binary CAT data.

I am hoping that users will be able to configure piWebCAT for FT1000 etc using:

- My documentation in the following pages on YAESU5 configuration.
- My FT920 configuration as a starting point / template

5.4 FT920 - Problems encountered

I add the short following section to make the reader aware of differences among these radios and difficulties with documentation.

Five byte commands

All the CAT manuals refer to a single byte (hex) **Opcode** and four parameters **P1**, **P2**, **P3** and **P4**.

For the **FT818** and **FT847** they are tabled in the order P1 P2 P3 P4 Opcode.
They must be transmitted in the order: P1 P2 P3 P4 Opcode

For the **FT920**, they are tabled in the order: Opcode P1 P2 P3 P4
They must be transmitted in the order P4 P3 P2 P1 Opcode
I had huge difficulty with this until I looked at the FT100MKV manual which is much clearer on the subject!

Example: Mode setting command:
The FT818 manual specifies: P1 ** ** ** 07 where 07 is the opcode byte (and * means don't care)
P1 is the mode selecting code, eg: 02 = CW 08 = FM etc

I configure as tu00000007
ie tu 00 00 00 07 where the mode code from the button is mapped into digits t and u.
The bytes are observed on scope decode to go out to the radio in the order:
P1 P2 P3 P4 Opcode ie 02 00 00 00 07 (for CW)

The FT920 manual specifies: 0C P1 ** ** ** where 0C is the Opcode and P1 is the mode code.
(mode codes are totally different from the FT818 and include separate codes for VFO A and VFO B.)
The bytes are actually sent in the order P4 P3 P2 P1 Opcode
and so I must configure as: 000000xx0C (I use xx to allow hex code mapping)

Received frequency data

To read in the 2 x 14 bytes VFO frequency, mode etc we use 00 00 00 03 10 (configured as 0000000310)

The manual says VFO A frequency is in bytes 2 to 5. - In piWebCAT we are zero-indexed and specify 1 to 4.

The **answermask** is **#28:01:04:FF:bbbbbbbb** 28 bytes to receive, extract 04 bytes starting at byte 01
The bytes are mapped to the following lower case letters: bbbbbbbb.
(The FF bit mask is unused as it is only for single byte extraction)

I found the download manual to be unclear on the format of the received frequency.
It s actually in binary, ie 14.123450 Mhz is1412345 (to 10Hz) and is received from the radio as 00158CF9.
... I discovered this - not from the manual, but from my Siglent scope's serial data decode facility

I have to use bbbbbbbb to force conversion from binary to decimal. (BCD would use xxxxxxxx)
Note that:
Frequency ***setting*** for the FT920 is in BCD (binary coded decimal) .. not binary as frequency read above.
Frequency ***setting*** and ***reading*** for the FT100MP MkV are both in BCD (The manual is MUCH better)
... all perfectly manageable once you know the correct formats.

FT920 - recall VFO command

This is sent as 00 00 00 P1 05 where, from the manual:
P1 = 00 is VFOA and P1 = 01 is VFOB
I may be missing something here, but my findings were :
P1 = 0 does nothing. P1 = 01 toggles between VFOA and VFO B
This means that I therefore provide a Swap A/B button but no VFO buttons.
The Swap buttons toggles between the two VFO datasets in VFO A.
I do not use the allocated Swap button because this has built inVFO A/B functions which require configuration access to specifically VFO A and VFO B, and this doesn't appear to be available.

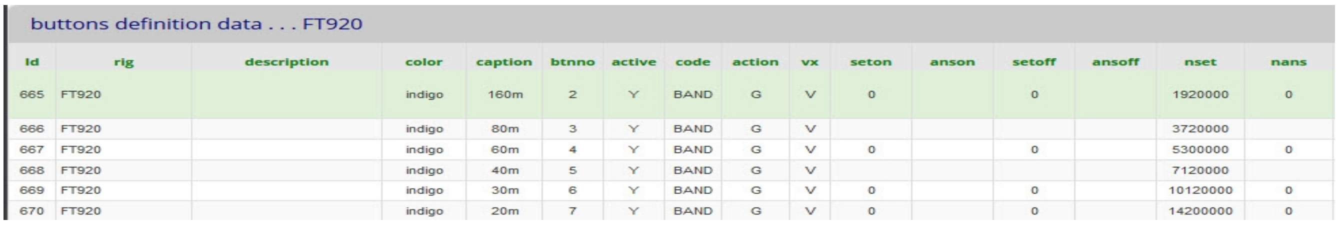
See - IMPORTANT special buttons

FT920 Split on/off:

The manual says send as 00 00 00 P1 01 where P1 = 00 is OFF and P1 = 01 is ON.
My findings were that this switches Tx between VFOA and VFOB
(which is not exactly split on/off !!)

FT920 band switching.

There appears to be no band switching command.
The only way to use the band switching buttons was to configure each with a start frequency for each band as shown:



buttons definition data . . . FT920

Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset	nans
665	FT920		indigo	160m	2	Y	BAND	G	V	0		0		1920000	0
666	FT920		indigo	80m	3	Y	BAND	G	V					3720000	
667	FT920		indigo	60m	4	Y	BAND	G	V	0		0		5300000	0
668	FT920		indigo	40m	5	Y	BAND	G	V					7120000	
669	FT920		indigo	30m	6	Y	BAND	G	V	0		0		10120000	0
670	FT920		indigo	20m	7	Y	BAND	G	V	0		0		14200000	0

Band buttons are a group and so use fields **nset** and **nans** to send data and to match against received data.

We still link to **catcodes** with **code = BAND**.   The data in field **nset** is the start frequency for the band (NOT a band code)
catcodes is configured to send the data (eg: 3720000) as a frequency   (because, as stated above - there is no band command?)
piWebCAT quickly responds to the frequency change by recognising the new band and changing its tuning scale.

There was a problem here: If you change band by just changing frequency via CAT, no mode change occurs.
The FT920 continues with the operating mode of the the old band and in fact adopts the mode for the new band.
(The radio does in fact store a mode for each band, and if you change bands with the radio's band switches,
the stored mode for the new band is applied)

piWebCAT's solution to the band change problem:

Frequency:
When a band is first selected (in a piWebCAT session), the frequency will be from in the button's table above.
piWebCAT is repetitively reading the FT920 frequency.
At each read, I check the frequency against band limits and then store it for the current band.
So piWebCAT has a record of the latest frequency for each band (modified on the radio or by piWebCAT's tuners.)

If we leave this band and then return to it, piWebCAT examines the **nset** field.
If this is > 10000, then this indicates that we are configured to change band by frequency (not by band code).
piWebCAT uses the stored latest band frequency to change band rather than the configured start frequency.

Mode
Mode is repetitively read from the radio and is stored by piWebCAT for each band.
I have to lookup the **nset** value rather than use the **nans** (which matches received data)
because they are different for the FT920.

On band change, if we are using piWebCAT's stored band frequency as described above,
then we use piWebCAT's stored band mode.
This mode is sent to the radio, 2 seconds after sending the frequency.

5.5 FT1000MkV   and other Yaesu 5 - bytes command radios

I do not possess an FT1000 MkV and do not have access to one.

I have examined the MkV manual.
The CAT section is better written than that of the FT920 and has more detailed explanations.

The MkV has a much larger range of CAT commands than the FT920.
In particular, there are write commands for a wide range of EDSP parameters.

However, unless I have missed something, there appears to have no band change command.
If this is correct, then the above discussed FT920 solutions will be needed.

I am happy to closely collaborate with anyone wishing to set up piWebCAT for these radios.
... either by direct email or via the iogroup.

If a good configuration can be produced, then we can make it available to others.
(It is possible, using MySQL Front, to extract a radios's configuration from your database as an SQL file.
The files can them be modified into a form which will simply add to another user's database)

5.6 YAESU5 configuration - frequency - FT818 example

You need to first look at the documentation on ASCII command masks.

The outgoing commands for all YAESU5 radios are five bytes:

- four bytes of optional bcd (binary coded decimal) data
- following by a hexadecimal opcode byte.
- The **readbytes** and **setbytes** fields are ***always*** set to 10 (ie: 10 chars generating 5 bytes)

Set frequency - FT818 example

The manual specifies the frequency in four bytes: P1 P2 P3 P4 followed by opcode = 01

(Be aware that the FT920 manual notation is reversed, ie: P4 P3 P2 P1 Opcode)

Thus: 01 42 34 56 01 will set 14.234560 MHz (frequency resolution = 10 Hz)

We configure this in the **catcodes** table as: **setmask = gfedcmht01**
ie: ten ascii characters which will be translated into five bytes.

In detail: gf ed cd ht 01. t is replaced by tens, h by hundreds, m by thousands

(Note that there is no **u = units** because the resolution is 10Hz)

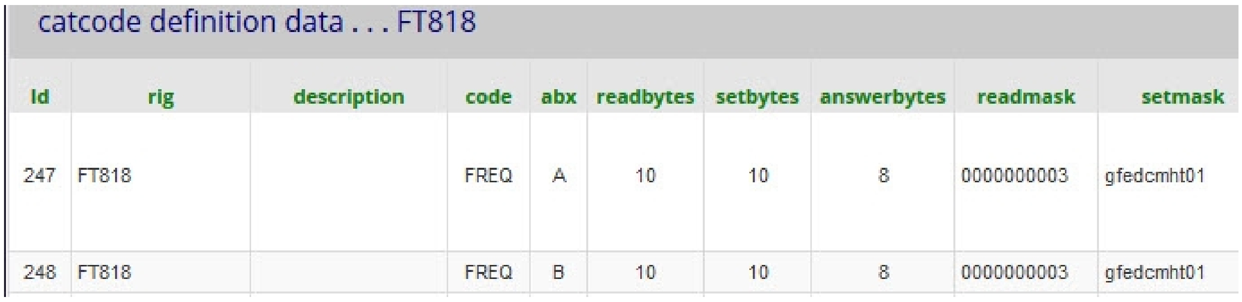
With YAESU5, the setbytes field is ALWAYS set to 10.

14.234560 MHz from piWebCAT's tuner will result in a ten char ascii string = '0142345601'
The server (with YAESU5 selected) will convert this to five bytes 01 42 34 56 01

Changing band
Note that The YAESU5 radios appear to have no band selecting command.
So we also have to use frequency setting to change band with the user's choice of initial frequency.

Below are the frequency configurations in the catcodes table.
Frequency setting will be driven from:

- piWebCAT's tuner OR
- the band sector buttons configured as a group in the buttons table.

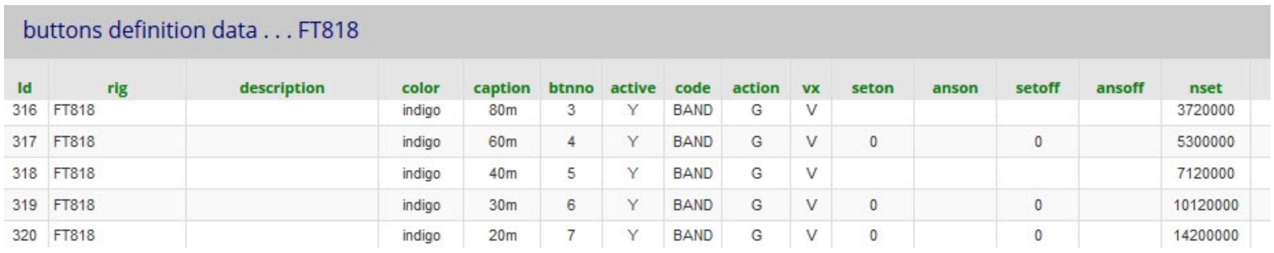


catcode definition data . . . FT818

Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask
247	FT818		FREQ	A	10	10	8	0000000003	gfedcmht01
248	FT818		FREQ	B	10	10	8	0000000003	gfedcmht01

Note that for tuner operation, the frequency read message to the server are internally generated from the tuner.
They must use code = FREQ in the catcodes table (which defines the server response)

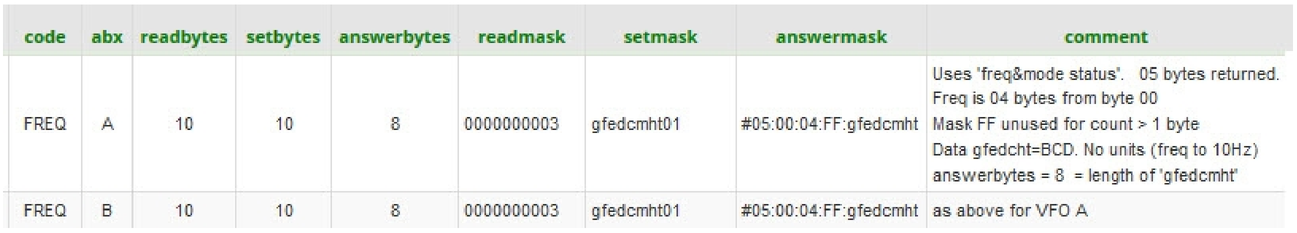
*For **band changing**, the FREQ commands to the server are generated by the grouped BAND buttons.*
*At the server, they are handled by the same **catcodes** records as the tuner generated commands.*



buttons definition data . . . FT818

Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset
316	FT818		indigo	80m	3	Y	BAND	G	V					3720000
317	FT818		indigo	60m	4	Y	BAND	G	V	0		0		5300000
318	FT818		indigo	40m	5	Y	BAND	G	V					7120000
319	FT818		indigo	30m	6	Y	BAND	G	V	0		0		10120000
320	FT818		indigo	20m	7	Y	BAND	G	V	0		0		14200000

Read frequency (and mode)   FT818 example



code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask	comment
FREQ	A	10	10	8	0000000003	gfedcmht01	#05:00:04:FF:gfedcmht	Uses 'freq&mode status'. 05 bytes returned. Freq is 04 bytes from byte 00 Mask FF unused for count > 1 byte Data gfedcht=BCD. No units (freq to 10Hz) answerbytes = 8 = length of 'gfedcmht'
FREQ	B	10	10	8	0000000003	gfedcmht01	#05:00:04:FF:gfedcmht	as above for VFO A

The above image shows the **readmask** for frequency reading as **0000000003**.

The manual specifies five bytes: * *   * * 03   (where * =   don't care)

So we enter ten characters **0000000003**
   These ten characters are transmitted to the server and translate to bytes 00 00 00 00 03.

The ten characters are sent to the server verbatim:
 - as is any character that is not in the reserved lower case mask set: g f e d c m h t u s x

Handling data frequency data read from the radio - *This is relevant to all YAESU5 data reads*

The FT920 frequency read response is 28 bytes. If this were handled in one operation the answermask would be 56 characters in length on the basis of two characters per byte. We don't do this !

We therefore use an answermask format of the form **#BB:SS:CC:MM:gfedcmht**   (no u = units here)

- **BB** is the total size of the bytes block to be read (The server needs this - it will wait for them or timeout)
- **SS** is the start position in the block of our data (zero indexed)
- **CC** in the count = size of our extract from the block
- **MM** is a hexadecimal bit mask which may be applied to single byte data.
- **gfedcmht**   is the actual mask to transform the CC bytes into 2 x CC characters

FT818 frequency read   **-** we extract 4 bytes from a 5 byte received data block

CC= 04, and the mask is **gfedcmht**. The server will interpret the four bytes as ***binary coded decimal data*** and transform them into a matching 8 character decimal string. eg: 02 81 23 45 >   28.123.450 MHz.

FT920 frequency read   - see below - we extract 4 bytes from a 28 byte received data block.

CC= 04, and the mask is **bbbbbbbb**. The server will interpret the four byes as ***binary data*** and perform a binary to decimal conversion, eg: 00 2A E9 B9   >   2812345 >   28.123.450 MHz

5.7 YAESU5 configuration - frequency - FT920 example

You need to first look at the documentation on ASCII command masks.

The outgoing commands for all YAESU5 radios are five bytes:

- four bytes of optional bcd (binary coded decimal) data
- following by a hexadecimal opcode byte.
- The **readbytes** and **setbytes** fields are ***always*** set to 10 (ie: 10 chars generating 5 bytes)

Set frequency - FT920 example

The manual specifies the frequency in four bytes: **P4 P3 P2 P1** followed by opcode **= 0A**

(Be aware that the FT818 manual notation is reversed, ie: P1 P2 P3 P4 Opcode)

Thus: **56 34 42 01** will set **14.234560** MHz (frequency resolution = 10 Hz)

We configure this in the **catcodes** table as: **setmask = htcmedgf01**
ie: ten ascii characters which will be translated into five bytes.

In detail: ht cm ed gf 01. t is replaced by tens, h by hundreds, m by thousands
c by ten thousands d by hundred thousands
e by millions, f by ten millions g by hundred millions

(Note that their is no u = units because the resolution is 10Hz)

All a bit tricky ... but that's how it is! .... It works fine !

With YAESU5, the setbytes field is ALWAYS set to 10.

Read frequency FT920 example

The command ('Status update') specified as; Opcoded = 10 then P1 ** ** P4

P1 = 03 returns 2 x14 bytes of data for:
VFOA in bytes 0 -1 3 (specified as 1 to E)
VFOB in bytes 14 - 27 (specified as 1 to E)

P4 is unused here with P1 = 03;

Within each 14 byte VFO block (I use 0 to 13 here, rather than the stated 1 to 14):

- byte 0 **Band selection** ?? what is it .. no explanation
- bytes 1 to 4 **Operating frequency** - Discovered to be in binary (not BCD)
- bytes 5 and 6 **Clarifier offset** - binary 2s complement
- byte 7 **Mode** data
- bytes 8 **Flag .. clarifier on/off, antenna selection etc**
- **bytes 9 and 10 Filter data**
- bytes 11 **and 12 CTCSS decoder data**
- **byte 13 Memory recall** data

Therefore, for VFOA, we need bytes 1 to 4 of the first 14 byte block
(= bytes 1 to 4 of the combined 28 byte block.)

We set the **answermask** to: **#28:04:01:FF:bbbbbbbb**

This means:

- Receive **28** bytes (we have to inform the USB/serial code how many bytes to wait for.)
- Extract **04** bytes starting at bytes **01**
- Ignore the **FF** -- This is a mask to extract specific bits ... only for single byte extract.
- **bbbbbbbb** - translate the four bytes into 8 characters as a hexadecimal (binary) string.

The corresponding mask for VFOB is #28:15:01:FF:bbbbbbbb
ie: The same, but starting at byte 1 of the second 14 byte half of the block = 14 + 1 = 15.

We must set the **answerbytes** field to **8** = no of chars in bbbbbbbb.

Explanation:

This system is designed to fit into the existing character based system (ASCII option)
All the interpretation is done on the server and returned to the web browser as a numeric string.
(ie: the data is transmitted back with the value part of the returned data array = 14234560)

This is all perhaps a little complex - but hopefully, the examples for F818, FT847 and FT920 with assist in applying a configuration to other radios (with help from the iogroup if needed)

The actual answermask for data is the 8 characters: bbbbbbbb,
hence we set answerbytes = 8 (*NOT 28*)

Clarifier FT920 (Rx only)

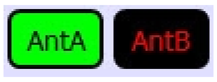
The clarifier is the one control where I can set up a slider with both on/off and reset buttons.
(and with two- way communication)

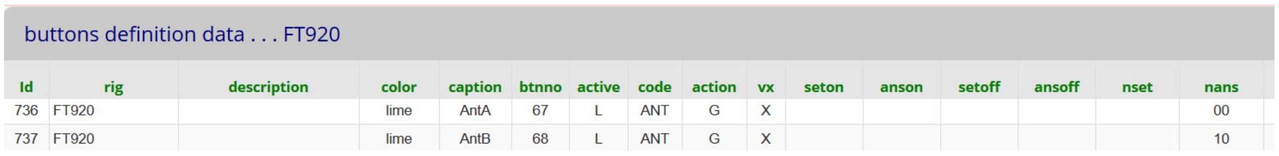
5.8 Example: FT920 antenna switching

The FT920 has three antenna connectors: Antenna A, Antenna B and Rx Antenna.
These are controlled by two buttons **Antenna A/B** and **Antenna Rx.**

I can find no way of controlling antenna selection from the CAT interface.

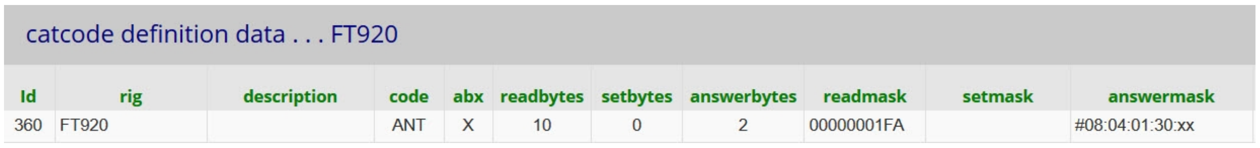
We can however read the the state of the antenna selection and display it on buttons configured as read-only 'LED' indicators (**active = L**)





buttons definition data . . . FT920

Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset	nans
736	FT920		lime	AntA	67	L	ANT	G	X						00
737	FT920		lime	AntB	68	L	ANT	G	X						10



catcode definition data . . . FT920

Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
360	FT920		ANT	X	10	0	2	00000001FA		#08:04:01:30:xx

I chose buttons 67 and 68 (left top row) and **code = ANT**.

The buttons are in a group of two (**action = G**)
(The **active = L** (LED indicator) facility works with single *toggling* buttons and with button *groups*.)

I set **vx = X** in buttons and **abx = X** in catcodes. This is appropriate for actions which are not per current VFO.
(See vx and abx)

Antenna selection is readable in the block of eight status bytes using command **00000001FA**.

The **answermask** is: **#08:04:01:30:xx** Receive **08** bytes, select **01** bytes at position **04** in the block.
Mask the byte with hex **30** to extract bits 4 and 5.

*Note that **answerbytes = 2** (One byte returned as **2** characters)*

From the manual: bit 4 = 0 for antenna A and bit 4 = 1 for antenna B. (result = hex 00 or hex 01)
bit 5 = 1 for Rx Antenna ... this overrides the indication of bit 4 (result = hex 20 or hex 30)

So we have:

Antenna A	result = 00
Antenna B	result = 01
Rx Antenna	result = 20 or 30

Unfortunately (without making a 'rigfix') piWebCAT cannot handle the 20 ***OR*** 30 result.

So I just have two buttons, A and B.
If Rx Antenna is selected then both the A and the B buttons are in the OFF state.

The values are in the **nans** field of the **buttons** table records: 00 for A and 10 for B.

Note that grouped buttons use the **nset** and **nans** fields, (Whereas toggling and single action buttons use **anson** / **ansoff** etc)

5.9 YAESU FT920 - Rx clarifier - on/off switching

You need to first look at the documentation on ASCII command masks.

The outgoing commands for all YAESU5 radios are five bytes:

- four bytes of optional bcd (binary coded decimal) data
- following by a hexadecimal opcode byte.
- The **readbytes** and **setbytes** fields are ***always*** set to 10 (ie: 10 chars generating 5 bytes)

The following examples illustrate on/off switching and use of a slider for clarifier tuning.

Note that piWebCAT is repeatedly reading different small amounts of data that are held in the common large data blocks. eg: mode, frequency, clarifier offset, flag (clar on/off) are all within the same 28 byte block. piWebCAT's design means that the web client has to read these separately for each task.
This is very inefficient and a waste of Rpi <> rig data bandwidth (at 4800 baud !)
I have therefore provided a caching system by which a single read store the data blocks in server SESSION variables. That the data is then available on the Rpi server without unnecessary extra reads of the same data. So for repetitive tasks, one of the read tasks does the read and the others use the cache. This is in an FT920 'rigfix'.

Clarifier controls

My example on the SD card uses button 18 and slider 18 (top right)
This has no reset-to default button.
If a reset button is required then this could easily be moved to the centre column.

Clarifier on/off button - FT920 example

The manual specifies clarifier setting in four bytes: **P4 P3 P2 P1** followed by opcode **= 09**

6	Clarifier Operations	09H	P1	P2	P3	P4	See Note 4

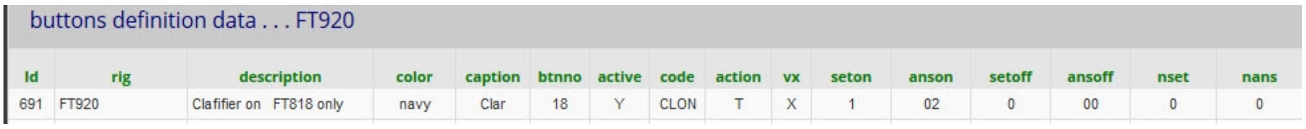
Setting clarifier on/off status.

The clarifier setting commands are illustrated above and in note 4 on the right.
They are used for both the button and the slider.
If P1 = FF, then P2, P3 and P4 control the slider.
For the button, we set P1 = 00 or 01 and we ignore P2, P3 and P4 (set them to zero)

<u>Note 4:</u>

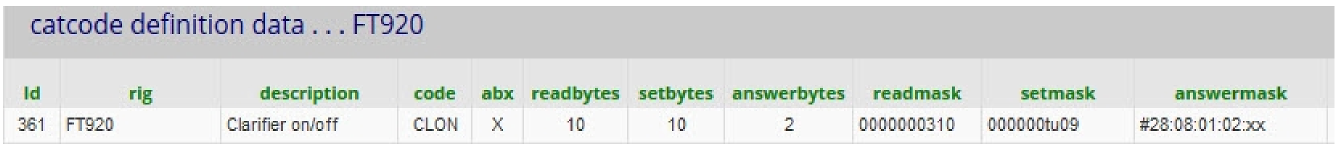
P1 = 00: RX Clarifier Off	P2 = 00: Clarifier Offset (+)
P1 = 01: RX Clarifier On	P2 = FF: Clarifier Offset (-)
P1 = 80: TX Clarifier Off	
P1 = 81: TX Clarifier On	P3 = 00~09 (kHz)
P1 = FF: Set Clarifier	P4 = 00~00 (100/10 Hz)

The buttons and **catcodes** table entries are:



buttons definition data . . . FT920

Id	rig	description	color	caption	btnno	active	code	action	vx	seton	anson	setoff	ansoff	nset	nans
691	FT920	Clafifier on FT818 only	navy	Clar	18	Y	CLON	T	X	1	02	0	00	0	0



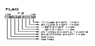
catcode definition data . . . FT920

Id	rig	description	code	abx	readbytes	setbytes	answerbytes	readmask	setmask	answermask
361	FT920	Clarifier on/off	CLON	X	10	10	2	0000000310	000000tu09	#28:08:01:02:xx

The **code = CLON** field links the **buttons** entry (client) to the **catcodes** entry. Button **action = T** (toggling).
seton = 1 and **setoff = 0** (from P1 = 01 for ON and P1 = 00 for OFF)

In the **catcodes** table, **setmask = 000000tu09** (00 00 00 tu 09 = ** ** ** P1 09)
t = tens **u** = units .
For the ON command, the server substitutes 01 for tu and sends **0000000109 .**
For the OFF command, the server substitutes 00 for tu and sends **0000000009 .**

Reading clarifier on/off status

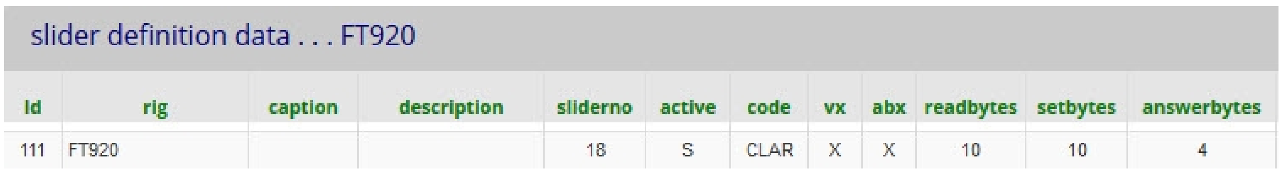


The Flag data bytes is byte 8 of the 14 byte block.
Bit 1 is Rx clarifier on/off.
To extract the bit we perform a logical AND with 02 hex.
This then returns 02 for ON and 00 for OFF.

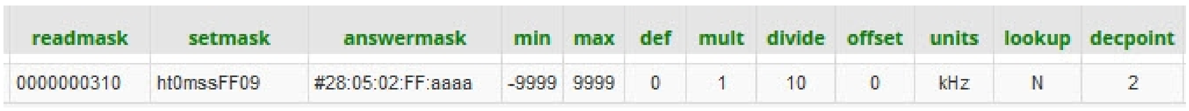
The answer mask is set to: **#28:08:01:02:xx**.
We have to receive all **28** bytes. We extract **01** byte at position **01** and mask it with **02** hex.
The result is returned as a two character number 00 or 02.
So in the **buttons** table, we set **anson = 02** and **ansoff = 00**.

5.10 Yaesu FT920 Clarifier tuning - slider example

The **sliders** table data is shown below - split into two parts because of its length.
(The **sliders** table contains client and server data
.... whereas for buttons, they are separate in **buttons** and **catcodes**)


slider definition data . . . FT920

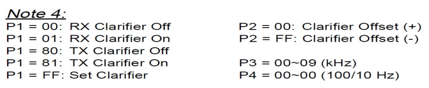
Id	rig	caption	description	sliderno	active	code	vx	abx	readbytes	setbytes	answerbytes
111	FT920			18	S	CLAR	X	X	10	10	4



readmask	setmask	answermask	min	max	def	mult	divide	offset	units	lookup	decpoint
0000000310	ht0mssFF09	#28:05:02:FF:aaaa	-9999	9999	0	1	10	0	kHz	N	2

Setting the clarifier offset by slider positioning

The clarifier setting commands are illustrated
in note 4 on the right.
They are used for both the button and the slider.
If P1 = FF, then P2, P3 and P4 control the slider.


Note 4:

P1 = 00: RX Clarifier Off	P2 = 00: Clarifier Offset (+)
P1 = 01: RX Clarifier On	P2 = FF: Clarifier Offset (-)
P1 = 80: TX Clarifier Off	
P1 = 81: TX Clarifier On	P3 = 00~09 (kHz)
P1 = FF: Set Clarifier	P4 = 00~00 (100/10 Hz)

P2 is FF for a negative offset and 00 for a positive offset. P3 and P4 are the frequency in Hz x10.

eg: to set - 5.230 kHz, the command would be in BCD (binary coded decimal) :

P4 P3 P2 P1 Opcode which is 23 05 FF FF 09

The **setmask** is **ht0mssFF09**

The slider + code generates -5230 which is sent to the server.

Using setmask, the server replaces h, t and m with 2, 3 and 5.
It replaces s with - because the value is negative (If positive then s replace by +)

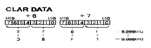
So far, this is standard ASCII behaviour as used for negative settings with Kenwood and later Yaesu radios.

So we have a data string **2305--FF09** (ie: two minus signs in the middle).

Finally, a **rigfix** changes -- to FF (++ to 00). **rigfix** is a field in the **rigs** table (For the FT920 we set **rigfix = FT920**)

Reading back the clarifier offset to position the slider.

Offset is read back as bytes 05 and 06 of the 14 byte VFO block.
(Labelled as bytes 6 and 7 in the manual)
The illustration in the manual shown right does not make the format clear. It is in fact a two's complement binary number (four hexadecimal digits, where FFFF = -1 FFFE = -2 etc)



We extract the two bytes and use mask aaaa which interprets as two-s complement binary and converts to a signed decimal number.

The **answermask** is : **#28:05:02:FF:aaaa**   **answerbytes = 4** because the conversion mask is **aaaa**

ie: read **28** bytes. Extract **02** bytes starting at byte **05** and convert as 16 bit (4 hex digits) 2s complement.

The number returned to the client as a numeric: -9999 to +9999.   (Hz x 10)

The **mult**, **divide**, **decpoint** and **units** fields control only the displayed value (not the slider)
I set **divide** to 10 to give -999 to +999 and then set **decpoint** to 2 to display as -9.99 to + 9.99 kHz.

Sync

I my database, I changed to **active = S** (auto sync) for the clarifier slider and on/off button.
This makes these controls follow the corresponding status on the radios (as well as the radio following piWebCAT)
See: Indicator controls. 'LED' option

In summary:

Unlike the modern Yaesu radios with easily constructed text commands, the older radios are difficult.
The set and read commands systems are totally different.
The read data has to be extracted from large multi-function data blocks.

5.11 FT920 and FT818 rigfix

Rigfix

If there are a few commands that cannot be implemented on certain radios, then a rigfix might be needed.
A rigfix is hard coded in PHP server code. A rigfix is applied to a radio by the **rigfix** field in the rigs table.
It has drop down selector data entry to select from the currently available options.

Only one rigfix option is selectable per radio. Rigfixes may share common PHP server code sections.
Rigfixes are named by the name of a radio (eg: FT920) but may be applied to other radios with the same command structure.

FT920 rigfix - caching of frequency and status reads

Frequency, mode, clarifier

Frequency, mode, clarifier offset and flag (for clar on/off) are read in a 2 x 14 bytes block for the two VFOs.
See: Example FT920 frequency

piWebCAT is structured to read one control items at a time.
This results in the same 28 byte block being read repeatedly to extract the different data items.
The FT920 only has one serial baudrate - 4800 baud and so frequent repetitive reads of 28 bytes potentially compromise its data bandwidth because of the repetitive tasks
Time periods, per rig are in the timings table. My current FT920 settings are:

• fmain	200ms	Read radio's main VFO (to follow the radios and for band change)
• fsub	600ms	Read the radio's sub VFO
• chkmode	2000ms	Check mode on radio
• sync	300ms	Clarifier on/off button has active = S so has repetitive updates from the flag byte. The clarifier slider has active = S so has repetitive updates from the 28 bytes block.

With **rigfix = FT920** selected:
Reading of VFO A (fmain ..200ms) reads the 28 byte block, uses it, and stores it in a SESSION variable.
All other repetitive reads from the 28 byte block use the SESSION variable copy and so do not need to access the radio,

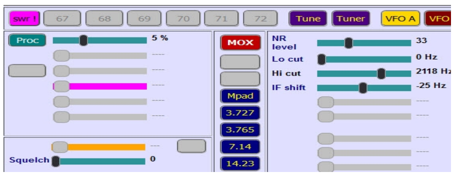
Server SESSION variables.
Server PHP code is launched for each individual read or write task. When the task is complete, the PHP process dies and all associated internal data is lost. We therefore store data in a server SESSION variable which persists for the duration of the web browser session.

FT920 rigfix - caching of read-only sliders and SWR buttons

The six sliders have **active = L.**
This makes them follow the radio values with a period determined by the **sync** field in **timings**.

The SWR indicator button has **active = L** and **colour = fuchsia.**
It follows the high SWR indication on the radio changing from OFF = black to ON = fuchsia.

All such sliders and buttons are read in turn from a circular queue which steps every **sync** ms.
Here the queue has six sliders and one button.



The slider values are read only. The FT920 CAT systems does not provide for remotely setting them.
The values are read using Read Meter (Opcode F7) which is also used for S meter, power meter etc.
There are four options, each returning four bytes plus a padding byte (F7)
Caching occurs, where possible, within each byte in order to reduce the number of actual reads from the radio.

FT920 and FT818 rigfixes changing -- and ++ to FF and 00

This is used for setting clarifier offset. See: FT920 clarifier slider tuning

setmask = ht0mssFF09 The numeric value is **m h t** (thousands, hundreds, tens (no units)
ss is the sign (twice)

So the mask converts **-5230** into **2305--FF09**

The rigfix converts the **--** to **FF** (or ++ to 00) to send **2305FFFF09** to the radio.

Note that for the FT818 , the manual simply specifies non-zero rather than FF for negative.

FF is non-zero and so does the job !

6.1 piWebCAT - Icom CI-V configuration

piWebCAT's control actions are NOT read-modify-write (unlike my EncoderCAT project)
The controls are synchronised to the radio's settings at startup, on band change and by the reload button.
*This uses the **read** and **answer** configurations to request the data and interpret the answer.*
*Subsequent button or slider action commands use **set configurations** with the new position of a slider,*
*or from known buttons states. ie: only the **set** commands are used.... there is no feedback.*
(eg: for toggled buttons, piWebCAT uses the button's remembered on or off state to determine the new state.)

Icom CI-V uses CAT byte sequences that are specified in hexadecimal:

- **Hexadecimal** command codes, eg: 1A hex
- **Decimal data values** represented in **Binary Coded Decimal format** (BCD)

Hexadecimal bytes are represented in this document with the prefix 0x. (as used C and PHP languages)
eg: 0x2A is hexadecimal 2A = 2 x16 + 10 = decimal 42

BCD format represents two decimal digits as a single hexadecimal byte.
eg: decimal 37 is 0x37 ... easy to read (but on hex to dec conversion = 3 x16 +7 = 55 decimal)

piWebCAT software does the decimal to BCD conversion for you.
This allows you to enter command 050014 as such in decimal.
The value is sent to the server as decimal 050014.
The server then transmits this as three bytes: 0x05 0x00 and 0x14.

Icom's command system is complicated by what would appear to be historical / evolutionary features.
The range of formats includes:

- single hex command byte
- hex command byte + hex sub-command
- hex command byte + hex sub-command byte + bcd data bytes. (read only - but read/write in later rigs)
- hex command byte + two bcd data byes (read/write). The bcd bytes referred to as sub-command
- hex command byte + three bcd data bytes. (read/write). The bcd bytes referred to as sub-command

piWebCAT provides a CI-V user configuration system that supports these formats.

piWebCAT CIV tables

Three table are involved in CIV communication:

- **catcodesciv** Generating server to radio commands from **buttons** table request from the client.
  Decoding button state data from the radios at start up and on VFO or band change.
  Also has the CIV control information for frequency read and write.
- **slidersciv** One table providing client AND server data fields.
  (Unlike **buttonsciv** which controls only client functions and uses **catcodesciv** for server. functions)
- **meterciv** Controls client and server meter reading functions.

These three tables control the CIV communications using fields :
rigaddr, **rpiaddr**, **cmdtype**, **datadigits**, **com**, **subcom** and **subcom4or6**.

How this works is described below rather in the individual table descriptions.

CI-V control fields

I use the abbreviation **com** for command and **subcom** for sub command.

Hexadecimal bytes are shown as 0xFE etc. BCD was explained above.

CI-V commands have the form show below.

0xFE | 0xFE | rig addr. byte | control. addr. byte | com byte | subcom byte(s) | databyte(s) | 0xFD

The **0xFE** bytes are start / security bytes.   **0xFD** is an obligatory terminating bytes.

In *data returned from the radio*, rig addess. and controller address are reversed.

The **cmdtype** field in a table defines the type of CI-V command and how the other fields are used.

cmdtype has seven possible values which are described below.

The codes (eg: C_S_DATA) are descriptive.
eg: C_S6_DATA has a six digit decimal subcommand > coded into 3 BCD bytes
com is always a hexadecimal byte, eg 0x1A
subcom is a hexadecimal byte   OR   2 BCD bytes   OR 3 BCD bytes

The seven **cmdtype** values are:

• **C_S_DATA**	Com byte (hex)	Subcom byte (hex)	Data byte(s)
• **C_S6_DATA**	Com byte (hex)	3 Subcom bytes (BCD)	Data byte(s)
• **C_S4_DATA**	Com byte (hex)	2 Subcom bytes (BCD)	Data bytes(s)
• **C_S_ONOFF**	Com byte (hex)	Subcom byte (hex)	1 or 0 for on/off
• **C_S**	Com byte (hex)	Subcom (hex)	
• **C_ONOFF**	Com byte (hex)	1 or 0 for on/off	
• **C_DATA**	Com byte (hex)	Data byte(s)	
• **C_ONLY**	Com byte (hex)	no subcom, no data (eg: 0x0A = memory to VFO)	

The tables have CIV control fields as follows. (**cmdtype** dictates the use of the other fields.)

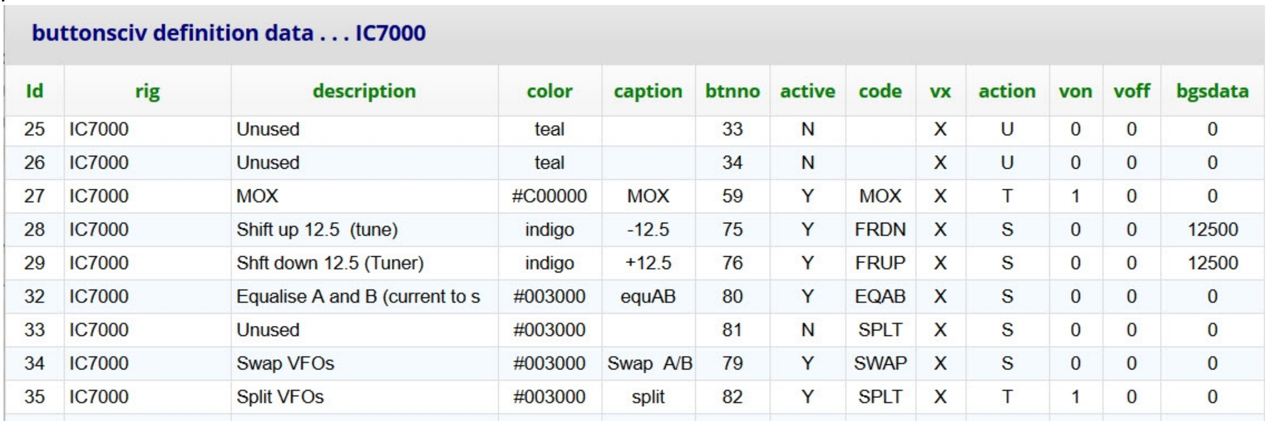
- **rigaddr**   The address of the radio - usually 0x70 .. but can be changed if multiple radios.
- **rpiaddr**   The send address of the piWebCAT RPi   I always use 0xE0
- **cmdtype**   as above
- **datadigits** The number of decimal digits. When configuring a command, look at the data.
  Example: if data is specified as 0-255 then datadigits = 3
  Two BCD bytes will be used. eg: 147 is coded as 0x01 0x47.
  (Care needed!! eg: If I set datadigits = 5 here, then 3 bytes are sent - and it fails)
- **com**   The hex com, eg 1A
- **subcom**   The hex subcom if used -- only for C_S_DATA, C_S_ONOFF and C_S.
- **subcon4or6** Decimal subcom - see above   - for C_S6_DATA and C_S4_DATA

6.2 piWebCAT - Database table - buttonsciv   See buttons configuration notes   timing.disable

- **rig** The current radio - drop down selector (from radios table)
  Must have same spelling through the tables.
- **description** Descriptive text- no function
- **colour** Button's background colour at startup.
  Can be a standard HTML color, eg teal,indigo etc or a numeric colour:
  ie: #RRGGBB eg #223344 where 0x22 is red level (0x00 - 0xFF)
- **caption** The caption to be applied to the button at start up. Must fit the button's width.
  Try something and then observe. (Lower case letters are MUCH narrower!)
- **btnno** The button's unique, fixed, numeric identifier. See: Button and slider numbering
- **active** Y = button active N = button inactive S = active + sync (repetitive state update from rig)
  L = sync and 'LED' read-only indicator lamps (See: LED buttons)
- **code** 3 or 4 upper case characters. This links a button command to its action
  on the server. It must match the **code** for the linked record in **catcodes**.
  See below for more explanation.
- **vx** **V** for client to send A or B according to current VFO. (A and B catcodes records)
  X to send X (single non-VFO dependent server record. **U** no action to server.
  (U = do not participate in buttons state store and retrieve) See: vx and abx
- **action** **S** = single momentary. Button flashes briefly. Sends **von** column value.
  T = toggled. Alternates on/off. Client code highlights it when on and remembers
  its current state. Sends **von** or **voff** column value.
  G = grouped. Is in a group of other **G** buttons *with the same **code**.*
  Only one of the group is highlighed. The data value associated with each button is in
  the **bgsdata** field (which is also used in setting button state from the radio at startup.
  M = a meter button. **R** = a slider reset buttons
- **von** ON value for an on/off button ie: action = **S** or **T** (usually 1)
- **voff** ON value for an on/off button ie: action = **S** or **T** (usually 0)
- **bgsdata** Used for grouped buttons data (ie: **action** = **G**). Each button in the group has an
  identifying code value in **bgsdata.**

The **buttonciv** table fields have to be set up in cooperation with the corresponding record(s)
in the **catcodesciv** table.

piWebCAT's **buttonsciv** table editor is shown below:



buttonsciv definition data . . . IC7000

Id	rig	description	color	caption	btnno	active	code	vx	action	von	voff	bgsdata
25	IC7000	Unused	teal		33	N		X	U	0	0	0
26	IC7000	Unused	teal		34	N		X	U	0	0	0
27	IC7000	MOX	#C00000	MOX	59	Y	MOX	X	T	1	0	0
28	IC7000	Shift up 12.5 (tune)	indigo	-12.5	75	Y	FRDN	X	S	0	0	12500
29	IC7000	Shft down 12.5 (Tuner)	indigo	+12.5	76	Y	FRUP	X	S	0	0	12500
32	IC7000	Equalise A and B (current to s	#003000	equAB	80	Y	EQAB	X	S	0	0	0
33	IC7000	Unused	#003000		81	N	SPLT	X	S	0	0	0
34	IC7000	Swap VFOs	#003000	Swap A/B	79	Y	SWAP	X	S	0	0	0
35	IC7000	Split VFOs	#003000	split	82	Y	SPLT	X	T	1	0	0

I choose to leave unused inactive entries in the **buttons** and **buttonsciv** tables,
They carry the unused, fixed, unique **btnno** values. See: Button and slider numbering

The shift up and down buttons must use codes FRUP and FRDN as this is a hard coded function on the server.
See: Frequency up /down buttons

Unmatched button conditions

If a button group does not have a button to match all corresponding states on the rig, then an error arises.
My FTdx101D has fifteen modes. If I only configure buttons for seven, then selecting one of the other eight
on the rig will cause a problem. This is discussed at the end of section 2.11 Learning guide Transceiver-H-A

Tx meter buttons

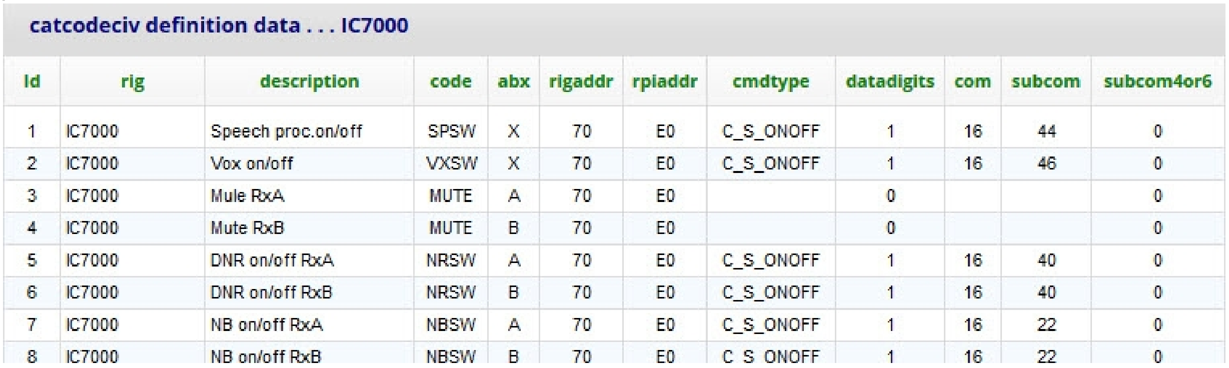
These do not communicate with the server - they select the Tx meter on the client
action **= M** and the code field is fixed as **TXME**
Clicking the button does not issue a command to the radio. It records the Tx meter selection on the client so that the repetitive meter read process read the correct meter.
TXME also identifies the group members in order to clear the group before highlighting the clicked button.
Meter reading codes are set up in the meter table.
The link to the meter table is the button numbers 61 to 65. See section 4.5 Table meter

6.3 piWebCAT - Database table - catcodesciv

catcodesciv is used on the server to communicate with the radio for buttons and frequency.
(None of its data is loaded to the client)

- **rig** The current radio - drop down selector (from radios table)
  Must have same spelling through the tables.
- **description** Descriptive text- no function
- **code** The link to the buttons table - use my defaults where possible.
  Note that for frequency reading and setting, the code must ***FREQ.***
- **abx** A or B if there is pair of entries one for each VFO (eg: Mute RxA, RxB)
  X if A or B not relevant. (eg:swap VFOs, Tuner etc) See: vx and abx
- **rigadr** 70 (hex) is the default on some Icon radios
- **rpiaddr** The controller address - use E0.
- **cmdtype** The Icom command type. Control data formatting. See CIV control fields
- **datadigits** The number of decimal digits in set pr answer data. See CIV control fields
- **com** The hexadecimal command byte.
- **subcom** The hexadecimal sub command byte. (if used)
- **subcon4or6** The decimal sub command (eg: 50014) when cmdtype = C_S6_DATA or C_S4_DATA.

piWebCAT editor for **catcodes** is shown below:



catcodeciv definition data . . . IC7000

Id	rig	description	code	abx	rigaddr	rpiaddr	cmdtype	datadigits	com	subcom	subcom4or6
1	IC7000	Speech proc.on/off	SPSW	X	70	E0	C_S_ONOFF	1	16	44	0
2	IC7000	Vox on/off	VXSW	X	70	E0	C_S_ONOFF	1	16	46	0
3	IC7000	Mule RxA	MUTE	A	70	E0		0			0
4	IC7000	Mute RxB	MUTE	B	70	E0		0			0
5	IC7000	DNR on/off RxA	NRSW	A	70	E0	C_S_ONOFF	1	16	40	0
6	IC7000	DNR on/off RxB	NRSW	B	70	E0	C_S_ONOFF	1	16	40	0
7	IC7000	NB on/off RxA	NBSW	A	70	E0	C_S_ONOFF	1	16	22	0
8	IC7000	NB on/off RxB	NBSW	B	70	E0	C S ONOFF	1	16	22	0

Note the **abx** column:
If there are separate commands for VFOA and VFOB, then there are two records: **abx** = A and **abx** = B.
Thus DNR on/off as two entries and speech proc on/off has one entry (so **abx** = X)

See CI_V configuration - examples

6.4 piWebCAT - Database table - slidersciv

See sliders configuration notes timing.disable

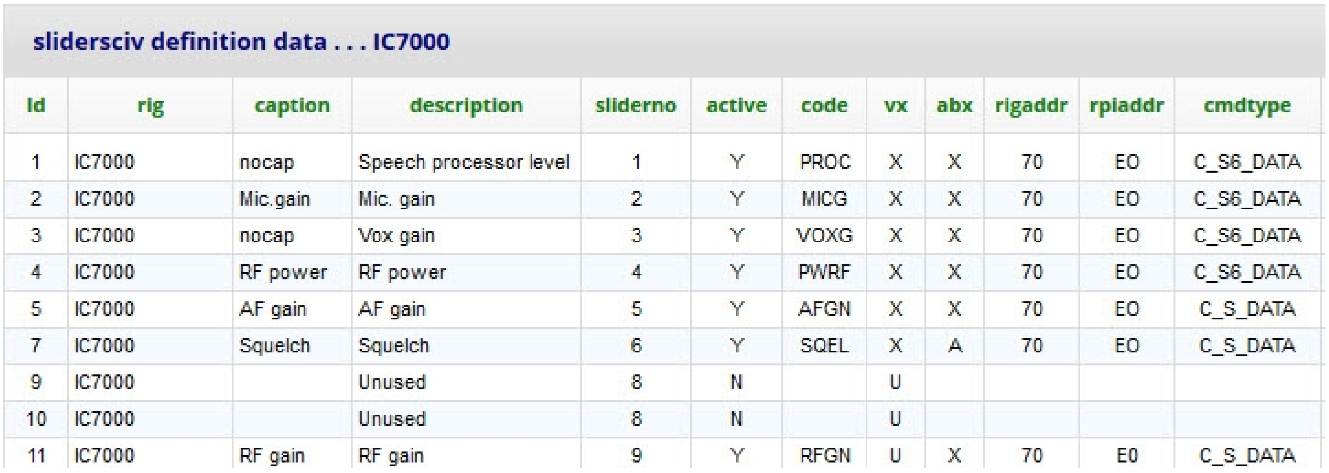
The configuration of **slidersciv** follows the same principles as for buttons.
The difference is that all *the client and server data is in one large table*
(where as **buttonsciv** referenced table **catcodesciv** for server functions)
So - at startup, piWebCAT extracts part of the data from **slidersciv** to store on the client.
The server, on receipt of a command, reads **slidersciv** from the database for server functions.

The client held data from the **slidersciv** table includes data for formatting of the numeric text displays of slider value. Slider values are set when read from the radio at startup and change on slider movement.

slider fields:

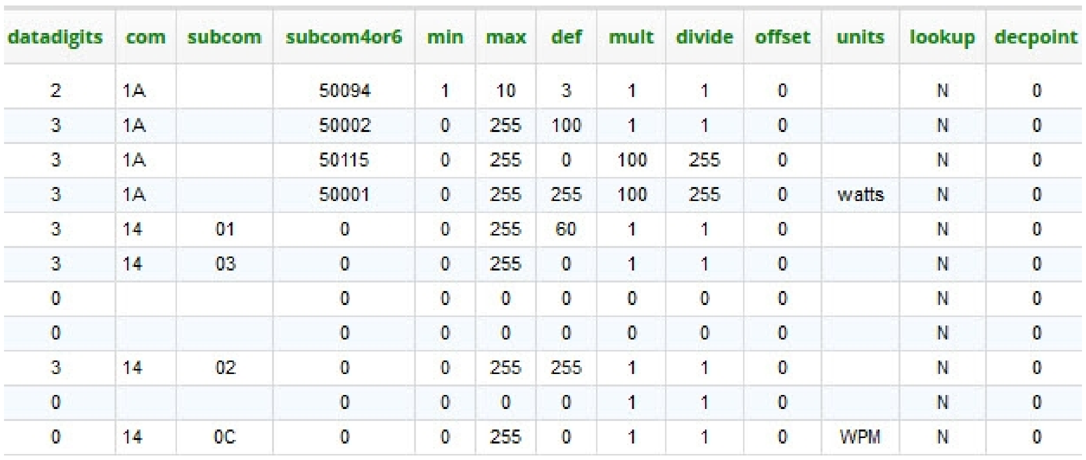
- **rig** The current radio - drop down selector (from radios table)
  Must have same spelling through the tables.
- **description** Descriptive text- no function
- **caption** The caption to the left of the slider. Note that this only applies to the central column of sliders with no adjacent on/off button with identifying caption.
- **sliderno** The sliders's unique, fixed, numeric identifier. See: Button and slider numbering
- **active** **Y** = slider active N = slider inactive (drop list selector)
- **code** 3 or 4 upper case characters. This links slider movement action on the client to its action processes on the server. It is transmitted to the server with data (jobdata)
  See below for more explanation.
- **vx** **vx = V, X or U**. Client field that controls storage of latest setting for each VFO.
  vx is set to **V** for dual receiver settings if the radio stores a different value per receiver.
- **abx** **A** or **B** if there is pair of entries one for each VFO eg: DNR level, RF gain
  X if A or B not relevant. (eg: RF power level, mic gain etc mode) See: vx and abx
- **rigadr** 70 (hex) is the default on some Icon radios
- **rpiaddr** The controller address - use E0. (Multiple controllers possible with different addresses)
- **cmdtype** The Icom command type. Control data formatting. See CIV control fields
- **datadigits** The number of decimal digits in set or answer data. See CIV control fields
- **com** The hexadecimal command byte.
- **subcom** The hexadecimal sub command byte. (if used)
- **subcon4or6 Th**e decimal sub command (eg: 50014) when **cmdtype** = C_S6_DATA or C_S4_DATA.
- **min** The minimum of the CAT data (ie: before scaling etc)
- **max** The maximum value of the CAT data.
  min and **max** ensure that the range of the sent CAT value spans the limits of the slider and that the slider response to radio initiated change also fits the range,
- **def** Default value - set by the associated reset button (not available on all sliders)
- **mult** Multiplier to scale CAT value for display.
- **divide** Divider to scale CAT value for display.
  eg: use (CAT * 100) / 255 to scale 0-255 to 0 - 100
  (Multiply first then divide).
- **offset** Offset applied to display value. eg: offset = -50 scales 0 -100 to -50 to +50.
- **units** Units after displayed value eg: Hz kHz etc
- **lookup** If lookup = **Y** then displayed value is taken from lookup table entries with matching code.
  Lookup uses the CAT value after scaling (if any).
  If lookup = **M** then display value from lookup table entry with matching **code** and lookup table **mode** field = current operating mode (eg: USB, LSB etc).
  For lookup = **M**, lookup table mode must be the mode button's caption -
  ie: LSB, USB, CW etc (NOT SSB) See Lookup table .
- **decpoint** Add decimal point = dec places. eg: decpoint = 3 for 3000Hz to 3.000kHz.

Below is a screen dump of the **sliderciv** table in the piWebCAT editor.
It is split into left and right parts.



sliderscív definition data . . . IC7000

Id	rig	caption	description	sliderno	active	code	vx	abx	rigaddr	rpiaddr	cmdtype
1	IC7000	nocap	Speech processor level	1	Y	PROC	X	X	70	EO	C_S6_DATA
2	IC7000	Mic.gain	Mic. gain	2	Y	MICG	X	X	70	EO	C_S6_DATA
3	IC7000	nocap	Vox gain	3	Y	VOXG	X	X	70	EO	C_S6_DATA
4	IC7000	RF power	RF power	4	Y	PWRF	X	X	70	EO	C_S6_DATA
5	IC7000	AF gain	AF gain	5	Y	AFGN	X	X	70	EO	C_S_DATA
7	IC7000	Squelch	Squelch	6	Y	SQEL	X	A	70	EO	C_S_DATA
9	IC7000		Unused	8	N		U				
10	IC7000		Unused	8	N		U				
11	IC7000	RF gain	RF gain	9	Y	RFGN	U	X	70	E0	C_S_DATA



datadigits	com	subcom	subcom4or6	min	max	def	mult	divide	offset	units	lookup	decpoint
2	1A		50094	1	10	3	1	1	0		N	0
3	1A		50002	0	255	100	1	1	0		N	0
3	1A		50115	0	255	0	100	255	0		N	0
3	1A		50001	0	255	255	100	255	0	watts	N	0
3	14	01	0	0	255	60	1	1	0		N	0
3	14	03	0	0	255	0	1	1	0		N	0
0			0	0	0	0	0	0	0		N	0
0			0	0	0	0	0	0	0		N	0
3	14	02	0	0	255	255	1	1	0		N	0
0			0	0	0	0	1	1	0		N	0
0	14	0C	0	0	255	0	1	1	0	WPM	N	0

See   CI-V configuration - examples

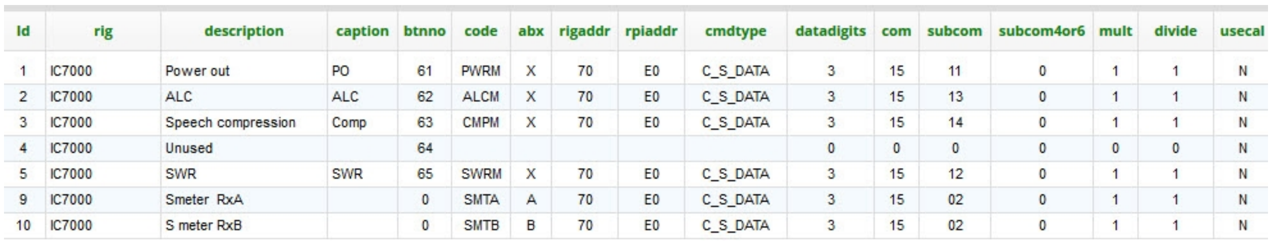
6.5 piWebCAT - Database table - meterciv

piWebCAT has an S meter on receive and five button-selected meter options in transmit.

The five Tx buttons are set up up the **buttonsciv** table
The **meterciv** table controls the subsequent repetitive meter reading.
The **timing** table sets the meter repetition interval.

The link between the **buttonsciv** table and the **meterciv** table is the **btnno** field values, 61 to 65.

The **meterciv** table is shown below for my IC7000



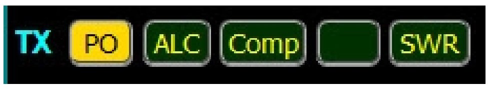
Id	rig	description	caption	btnno	code	abx	rigaddr	rpiaddr	cmdtype	datadigits	com	subcom	subcom4or6	mult	divide	usecal
1	IC7000	Power out	PO	61	PWRM	X	70	E0	C_S_DATA	3	15	11	0	1	1	N
2	IC7000	ALC	ALC	62	ALCM	X	70	E0	C_S_DATA	3	15	13	0	1	1	N
3	IC7000	Speech compression	Comp	63	CMPM	X	70	E0	C_S_DATA	3	15	14	0	1	1	N
4	IC7000	Unused		64						0	0	0	0	0	0	N
5	IC7000	SWR	SWR	65	SWRM	X	70	E0	C_S_DATA	3	15	12	0	1	1	N
9	IC7000	Smeter RxA		0	SMTA	A	70	E0	C_S_DATA	3	15	02	0	1	1	N
10	IC7000	S meter RxB		0	SMTB	B	70	E0	C_S_DATA	3	15	02	0	1	1	N

The are two S meter records (RxA and RxB) and four Tx meter records (IC7000 only has four Tx meters)
The two S meter entries have identical data control fields.

The S meter **code** fields **SMTA** and **SMTB** are ***fixed*** on the client and so must be entered as these so that the server will recognise them.
The Tx meter code fields are copied to the client at startup and so you can use any **code** of your choice.

piWebCAT has five Tx meter option buttons but the IC7000 has only four Tx values to display.
Button 64 is therefore unused. Button 64 is inactivated in the **buttonsciv** table.

Four of the Tx meter values have been assigned to four of the option buttons as shown below:



In the **meterciv** table, these are the items with **btnno** = 61, 62, 64 and 65. See Button and slider numbering

meterciv field list:

- **rig** The current radio - drop down selector (from radios table)
  Must have same spelling through the tables.
- **description** Descriptive text- no function
- **caption** The caption to apply to the button (overrides default TxmA, TxmB etc.)
- **btnno** The sliders's unique, fixed, numeric identifier as discussed earlier.
- **code** Specified once here. Used in message from client and recognised in server.
- **abx** If **A** or **B,** Client sends Vfo **A** or **B** to server. **abx** allows server to read appropriate meter.
  For Tx meters abx **= X** (not VFO-specific)
- **rigadr** 70 (hex) is the default on some Icon radios
- **rpiaddr** The controller address - use E0.
- **cmdtype** The Icom command type. Control data formatting. See CIV control fields
- **datadigits** The number of decimal digits in the answer data. See CIV control fields
  In meter table always 3 as meter CAT range is 0 255
- **com** The hexadecimal command byte.
- **subcom** The hexadecimal sub command byte (in IC7000 specifies the meter to be read.)
- **subcon4or6 Th**e decimal sub command (eg: 50014) when **cmdtype** = C_S6_DATA.
- **mult** Multiplier (default = 1)
- **divide** Divisor (default = 1) Meter CAT value is scaled by (CAT * mult) / divide
  before being optionally modified by the **metercal** table data.
- **usecal** **Y** or **N**. If **Y**, then the CAT value is processed by the calibration table.
  (20 point calibration for each meter with linear interpolation between points)

6.6 Icom CIV control - examples for the IC7000

Note that from **buttonsciv** table: Toggled buttons always send **von / voff.** Grouped always send bgsdata.

DNR on/off switching - single button

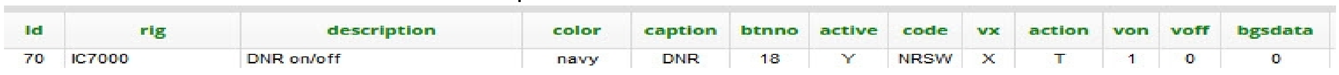
SIngle entry in buttonsciv table action = **T** (toggling) **von** = 1 **voff** = 0

- on click client transmits **code** = NRSW jobdata 1 or 0

Icom CAT manual states: **com** = 0x16 **subcom** = 0x40 data 1 = ON 0 = OFF

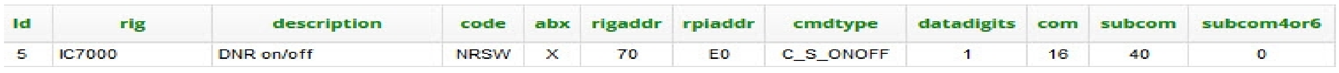
We therefore use **cmdtype** = C_S_ONOFF datadigits = 1 (the bcd data byte will be 0x01 or 0x00)

buttonsciv record vx = X = not VFO dependant



Id	rig	description	color	caption	btnno	active	code	vx	action	von	voff	bgsdata
70	IC7000	DNR on/off	navy	DNR	18	Y	NRSW	X	T	1	0	0

There is one **catcodesciv** record because there is a single setting which does not change with VFO change.
catcodesciv record



Id	rig	description	code	abx	rigaddr	rpiaddr	cmdtype	datadigits	com	subcom	subcom4or6
5	IC7000	DNR on/off	NRSW	X	70	E0	C_S_ONOFF	1	16	40	0

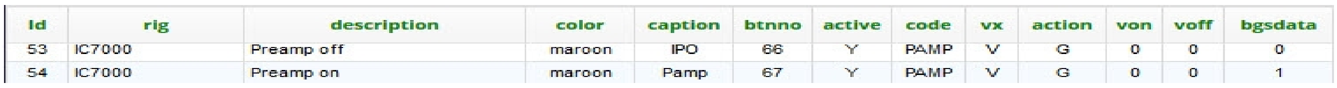
IPO / premp swiching - two buttons in a group of 2

There are two button records in **buttonsciv** because we have chosen to have a group of two:
pre-amp-off (IPO) and pre-amp-on
action = G (grouped) **code** = PAMP **bgsdata** = 0 (IPO) and 1 (preamp) -
bgsdata is used for grouped button values:
The CAT manual gives: com = 0x16 subcom = 0x02 preamp: 0 = off 1 = on.
For a pair of grouped buttons, we set bgsdata = 1 for the ON button and = 0 for the OFF button.
(we could have instead used a single toggling ON/OFF button where we would have set von = 1 and voff = 0)

We use **cmdtype** = C_S_DATA **datadigits** =1 (C_S_DATA means: com subcom data)

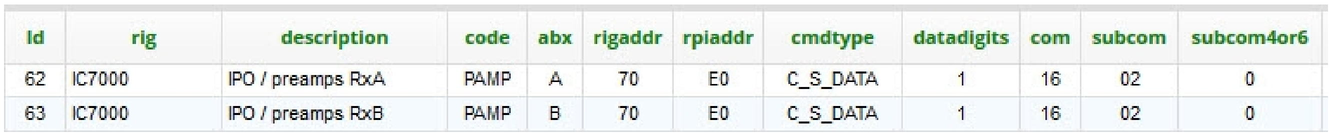
Note we could have used C_S_ONOFF here
- but I suggest C_S_DATA, particularly for larger groups of buttons with a range of **bgsdata** values.

buttonsciv records



Id	rig	description	color	caption	btnno	active	code	vx	action	von	voff	bgsdata
53	IC7000	Preamp off	maroon	IPO	66	Y	PAMP	V	G	0	0	0
54	IC7000	Preamp on	maroon	Pamp	67	Y	PAMP	V	G	0	0	1

catcodesciv records



Id	rig	description	code	abx	rigaddr	rpiaddr	cmdtype	datadigits	com	subcom	subcom4or6
62	IC7000	IPO / preamps RxA	PAMP	A	70	E0	C_S_DATA	1	16	02	0
63	IC7000	IPO / preamps RxB	PAMP	B	70	E0	C_S_DATA	1	16	02	0

Note that we have two catcodesciv records, one for the each VFO because the IC-7000 stores a different on/off status
for each VFO. The command is the same for the A and B records. The use of two records retains compatibility within the configuration system with radios that have a different command for VFO A and VFO B.

In **buttonsciv** (client) we set **vx = V.** This makes the client transmit A or B according to current VFO selection. It also makes the client store the latest setting for each VFO to avoid having to reload on VFO change

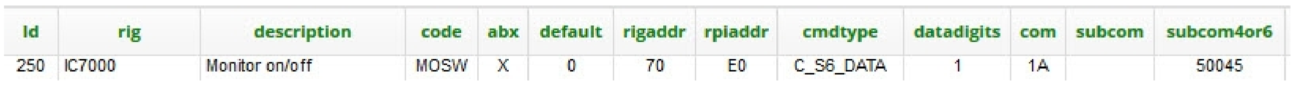
Monitor on/off switching - single buttons

Single button in table **buttonsciv** action = **T** (Toggled)) **code** = MOSW **von** =1 voff =0 vx = X

Client transmits code = MOSW jobdata = 1 or 0

Icom CAT manual states: com = 0x1A subcom = 50045 ... this will be BCD

We use cmdtype = C_S6_DATA com = 1A **subcom4or6** = 50046



Id	rig	description	code	abx	default	rigaddr	rpiaddr	cmdtype	datadigits	com	subcom	subcom4or6
250	IC7000	Monitor on/off	MOSW	X	0	70	E0	C_S6_DATA	1	1A		50045

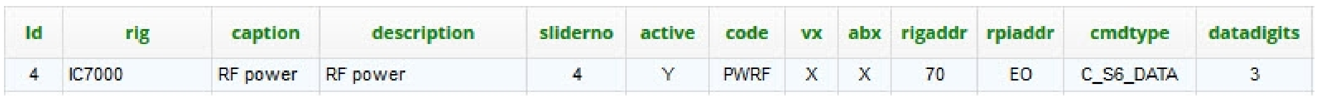
RF power level - slider

Slider data for client ***and*** server functions all in the **slidersciv** table.

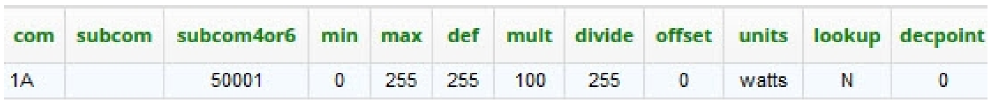
For **sliders**, there are no data fields in the table.
The table specifies **min** and **max**. These are always CAT value min and max (not displayed min and max)
When you move the slider, the value sent to the radio is based on **min** at slider left and max at slider right.
So here,min = 0 max = 255 (So, for example, at mid position the data to the radio is 128.)

Icom CAT manual states: com = 0x1A subcom = 50001 min = 0 max = 255

We use cmdtype = C_S6_DATA **com** = 0x1A **subcom4or6** = 50001 **datadigits** = 3 (because max = 255)



Id	rig	caption	description	sliderno	active	code	vx	abx	rigaddr	rpiaddr	cmdtype	datadigits
4	IC7000	RF power	RF power	4	Y	PWRF	X	X	70	EO	C_S6_DATA	3



com	subcom	subcom4or6	min	max	def	mult	divide	offset	units	lookup	decpoint
1A		50001	0	255	255	100	255	0	watts	N	0

Note: RF power is the only setting that is ***always*** loaded from the radios on VFO change (rather than use stored)
For this to happen, you must use code = PWRF.

Read write Frequency A

Not a slider nor a button but driven from the tuner.
Configured in the **catcodesciv** table

Frequency setting:
Mouse and thumbwheel actions modifiy variable **freqSet** in the client.
A timer driven task monitors **freqSet** and on change queues a message to the server.
This client activity is hard coded - there is no database configuration for it.

The control codes for the radios are configured in the **catcodesciv** table.
They use **code** = **FREQ** and **abx** = **A** for VFOA   and **code** = **FREQ** and **abx** = **B** for VFOB.
FREQ is hard coded in the client and so must not be changed.

Icom CAT manual states: **com = 0x05**   no subcom   **datadigits = 10**
(I could not find a statement on how many decimal digits
  - I used the Siglent scope to observe the returned data train on frequency read)

We use: **cmdtype** = **C_DATA**   **com** = **0x05**

Frequency read
Icom CAT manual states: **command = 0x03**
piWebCAT's configuration systems doesn't cope with separate **com** values for read and write.
However, all Icom radios appear to use set freq: **com = 0x05** and read freq: **com = 0**x03.
Therefore I was happy to hard code switching com = 0x05 to com 0x03 for a read.
A separate frequency reading entry is not therefore required. You just set up **com** = 0x05.

Operating mode is similarly coded (ie LSB, CW..)
Icom manual states: Select mode: **command = 0x06**.   Read mode: **command = 0x04**.
In the **catcodesciv** table, we use **com** = **0x06**, **cmdtype** = **C_DATA.**
The server code changes 0x06 to 0x04 for a mode read.

To summarise: for frequency and mode, we use the *setting* com value.
If the client requests read, the com value is automatically changed to the value for reading.

Note that there is a VFO A and a VFO B command in the **catcodesciv** table.
These are both the same except that one has **abx = A** and one has **abx = B**.
This is done for compatibility with radios that have different commands for the two VFOs.
(whereas Icom just have a command to tune the *current* VFO)

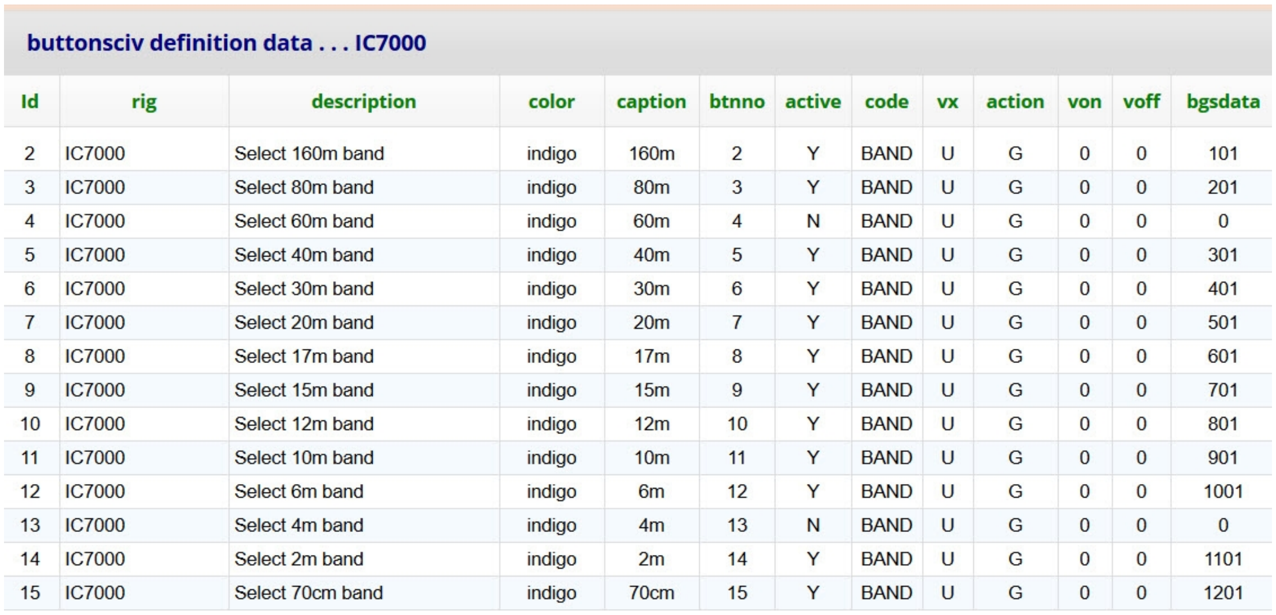
Band change buttons etc

The following is repeated in How it works.

I have a solution which works well and which is catered for by piWebCAT's standard CI-V configuration options.
The only workable command that I could assign to a band button was a *band-stacking register* command.
The band stacking register holds three preferred frequencies for each band:
- the most recently assigned, the second most recent and the third most recent.
For band button click, we configure in the database for piWebCAT to issue a ***write***
command to select the frequency of the 'most recent' register for the required band.

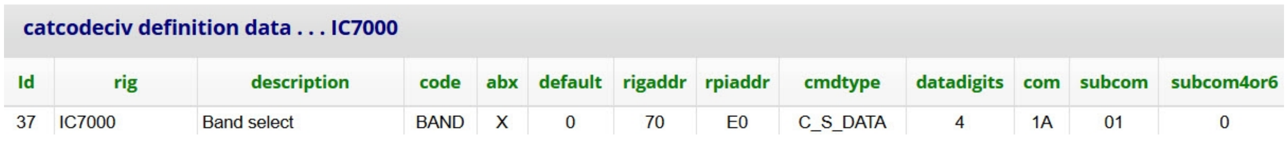
The command is **command = 0x1A**, **subcommand = 0x01** data = 4 bcd digits (2 bytes)
eg: data = 0301 is used for 40m ...the 03 is 40m the 01 specifies the 'latest'
So we configure **cmdtype** = C_S_DATA and **datadigits** = 4.

The band button sends the command to change frequency to the new band.
piWebCAT picks up the frequency change within 250ms and band change begins.



buttonsciv definition data . . . IC7000

Id	rig	description	color	caption	btnno	active	code	vx	action	von	voff	bgsdata
2	IC7000	Select 160m band	indigo	160m	2	Y	BAND	U	G	0	0	101
3	IC7000	Select 80m band	indigo	80m	3	Y	BAND	U	G	0	0	201
4	IC7000	Select 60m band	indigo	60m	4	N	BAND	U	G	0	0	0
5	IC7000	Select 40m band	indigo	40m	5	Y	BAND	U	G	0	0	301
6	IC7000	Select 30m band	indigo	30m	6	Y	BAND	U	G	0	0	401
7	IC7000	Select 20m band	indigo	20m	7	Y	BAND	U	G	0	0	501
8	IC7000	Select 17m band	indigo	17m	8	Y	BAND	U	G	0	0	601
9	IC7000	Select 15m band	indigo	15m	9	Y	BAND	U	G	0	0	701
10	IC7000	Select 12m band	indigo	12m	10	Y	BAND	U	G	0	0	801
11	IC7000	Select 10m band	indigo	10m	11	Y	BAND	U	G	0	0	901
12	IC7000	Select 6m band	indigo	6m	12	Y	BAND	U	G	0	0	1001
13	IC7000	Select 4m band	indigo	4m	13	N	BAND	U	G	0	0	0
14	IC7000	Select 2m band	indigo	2m	14	Y	BAND	U	G	0	0	1101
15	IC7000	Select 70cm band	indigo	70cm	15	Y	BAND	U	G	0	0	1201



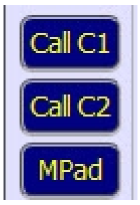
catcodeciv definition data . . . IC7000

Id	rig	description	code	abx	default	rigaddr	rpiaddr	cmdtype	datadigits	com	subcom	subcom4or6
37	IC7000	Band select	BAND	X	0	70	E0	C_S_DATA	4	1A	01	0

All band buttons send **code** = **BAND** and a **bgsdata** value of the form BBVV
where BB is band code are 1 to 12 and VV = 01 = the latest band stacking register entry.
abx = X because the command is not VFO specific on the IC7000.
(The command acts on the currently active VFO)

Memory selection buttons - examples

I configured 3 spare buttons in my IC7000 to manage the radio's memories.
The IC-7000 has five banks (A to E) each with 105 memory channels and also calling channels 0106 and 0107
The current IC-7610 has 100 memory channels. I could find no mention of 'banks' in the manual.
The buttons manage the use of memory channels that have been set up on the radio.
The parameters of each channel are set up on the radio, NOT by piWebCAT.



Call C1 selects memory channel 106. Then it sends a copy-memory-to-VFO command

Call C2 selects memory channel 107. Then it sends a copy-memory-to-VFO command.

MPad displays a numeric keypad for memory channel selection.
The keypad OK button selects a channel. Then it sends a copy-memory-to-VFO command.

Each button must be configured in **buttonsciv** (client settings) and in **catcodedsciv** (server / CAT settings)

Call C1 all C2 *(- no command-specific coding)*

These buttons selects calling channel C1 and C2 (memories 106 and 107 on the IC-7000)

Later Icom radios: IC-7610 CI-V uses same command (08 + channel number but with range 0001 to 0099)
No channel 106 - but these are single channel buttons with which you can use any memory channel.

The CI-V data shows **command = 08** (hex) followed by data 0106 or 107
Icom specify 0106 ie: 4 decimal digits so we must set **datadigits = 4** (which will send two BCD digits: 0x01 and 0x06)

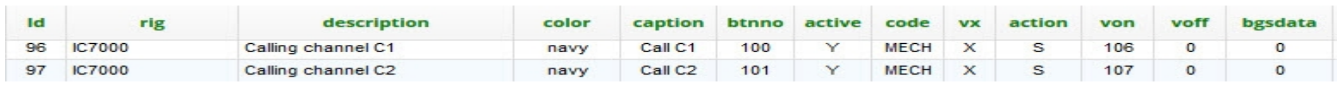
The two (or more) buttons share a common **catcodesciv** record.

The **buttonsciv** records control what happens on the client.
The **catcodedsciv** record controls what happens on the server ... ie: sending to the radio.
buttonsciv.action = 'S'. These are single click buttons (ie not grouped with others, not toggling on/off)
They are not VFO specific so I set **buttonsciv.vx = 'X'** **catcodesciv.abx = 'X'**
The data field for **buttonciv.action = 'S'** is **von** -- which is set to 106 or 107.
We must use **code = 'MECH' .** (Other code choices select the memory, but no following copy-memory-to-VFO occurs.)
cmdtype is set to C_DATA ie: command byte followed by some BCD data.

The database records for these memory select buttons are shown below.
The **MPAD** record is also shown for the keypad memory selection.

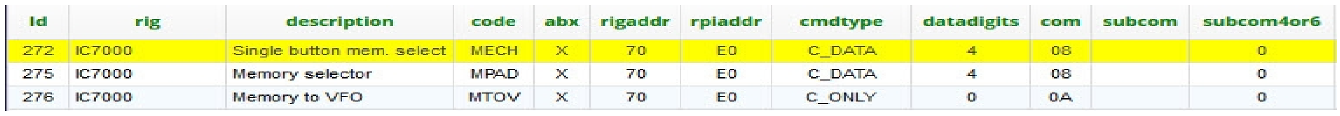
If **MTOV** is configured in **catcodesciv** as shown below, then MTOV will automatically apply the memory to the VFO.
If your radio automatically copies to VFO on memory selection, then do not configure MTOV.
(The command then is simply lost)

buttonsciv



Id	rig	description	color	caption	btnno	active	code	vx	action	von	voff	bgsdata
96	IC7000	Calling channel C1	navy	Call C1	100	Y	MECH	X	S	106	0	0
97	IC7000	Calling channel C2	navy	Call C2	101	Y	MECH	X	S	107	0	0

catcodesciv



Id	rig	description	code	abx	rigaddr	rpiaddr	cmdtype	datadigits	com	subcom	subcom4or6
272	IC7000	Single button mem. select	MECH	X	70	E0	C_DATA	4	08		0
275	IC7000	Memory selector	MPAD	X	70	E0	C_DATA	4	08		0
276	IC7000	Memory to VFO	MTOV	X	70	E0	C_ONLY	0	0A		0

Memory selector

The button is programmed to display a popup numeric key pad as a channel selector.
The code field must be set to **MPAD**, both in **buttonsciv** (to display the popup) and **catcodesciv** (to respond correctly) Note that is a special, hardcoded two-stage command system.
The button displays the popup.
The selected channel is sent client > server > radio when the OK button on the keypad is clicked.
One second later, a **MTOV** is automatically sent (no **buttonsciv** configuration needed for this)
If **MTOV** is configured in **catcodesciv** as shown below, then MTOV will automatically apply the memory to the VFO.

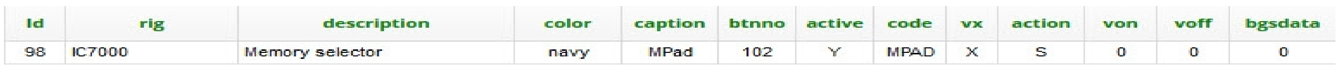
If your radio automatically copies to VFO on memory selection, then do not configure MTOV.
(The command then is simply lost)

buttonsciv: **action = 'S' code = 'MPAD'**   data fields von, **voff** and **btgsdata** are all unused.
(The data is the keypad selection)

The CI-V command is exactly the same as **'Call C1 / C2'** above,
ie: it sets a numeric channel (but from the keypad rather than fixed).

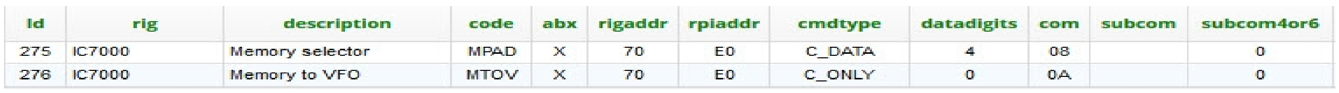
The database records for these memory select buttons are shown below.

buttonsciv



Id	rig	description	color	caption	btnno	active	code	vx	action	von	voff	bgsdata
98	IC7000	Memory selector	navy	MPad	102	Y	MPAD	X	S	0	0	0

catcodesciv

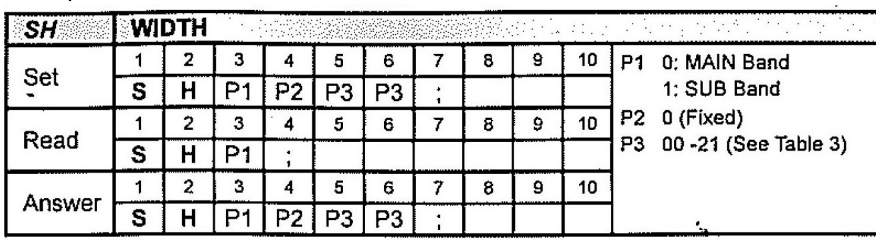


Id	rig	description	code	abx	rigaddr	rpiaddr	cmdtype	datadigits	com	subcom	subcom4or6
275	IC7000	Memory selector	MPAD	X	70	E0	C_DATA	4	08		0
276	IC7000	Memory to VFO	MTOV	X	70	E0	C_ONLY	0	0A		0

7.1 piWebCAT - Database table - lookup

The **lookup** table is used for the numeric value presentation from a slider when there is a non linear relationship between the CAT value and the displayed value.

Example from **Yaesu FTdx101D CAT manual**:



SH	WIDTH										
Set	1	2	3	4	5	6	7	8	9	10	P1 0: MAIN Band
	S	H	P1	P2	P3	P3	;				1: SUB Band
Read	1	2	3	4	5	6	7	8	9	10	P2 0 (Fixed)
	S	H	P1	;							P3 00 -21 (See Table 3)
Answer	1	2	3	4	5	6	7	8	9	10	
	S	H	P1	P2	P3	P3	;				

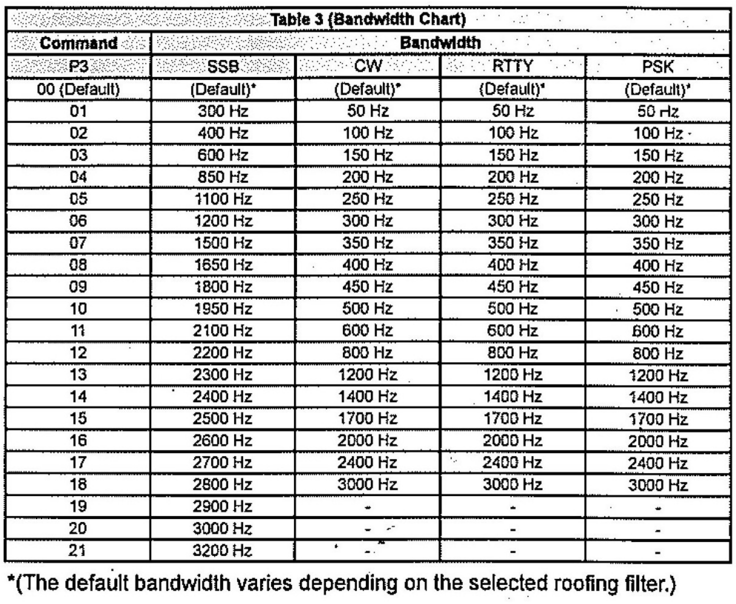


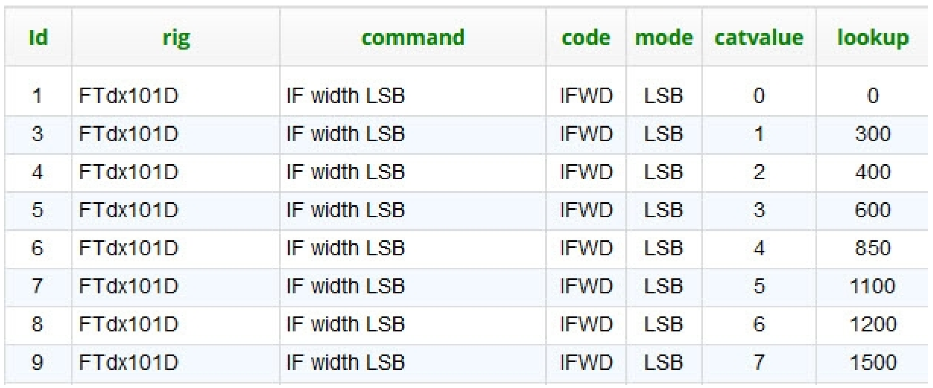
Table 3 (Bandwidth Chart)

Command	Bandwidth			
P3	SSB	CW	RTTY	PSK
00 (Default)	(Default)*	(Default)*	(Default)*	(Default)*
01	300 Hz	50 Hz	50 Hz	50 Hz
02	400 Hz	100 Hz	100 Hz	100 Hz
03	600 Hz	150 Hz	150 Hz	150 Hz
04	850 Hz	200 Hz	200 Hz	200 Hz
05	1100 Hz	250 Hz	250 Hz	250 Hz
06	1200 Hz	300 Hz	300 Hz	300 Hz
07	1500 Hz	350 Hz	350 Hz	350 Hz
08	1650 Hz	400 Hz	400 Hz	400 Hz
09	1800 Hz	450 Hz	450 Hz	450 Hz
10	1950 Hz	500 Hz	500 Hz	500 Hz
11	2100 Hz	600 Hz	600 Hz	600 Hz
12	2200 Hz	800 Hz	800 Hz	800 Hz
13	2300 Hz	1200 Hz	1200 Hz	1200 Hz
14	2400 Hz	1400 Hz	1400 Hz	1400 Hz
15	2500 Hz	1700 Hz	1700 Hz	1700 Hz
16	2600 Hz	2000 Hz	2000 Hz	2000 Hz
17	2700 Hz	2400 Hz	2400 Hz	2400 Hz
18	2800 Hz	3000 Hz	3000 Hz	3000 Hz
19	2900 Hz	-	-	-
20	3000 Hz	-	-	-
21	3200 Hz	-	-	-

*(The default bandwidth varies depending on the selected roofing filter.)

The Fdx101D has different values for SSB and the other modes.
The **slider** table **lookup** field can have values **N**, **Y** or **M.** (N = no lookup)
If **slider.lookup** = **Y,** then the lookup table entries must have **mode = Y**.
If **slider.lookup** = **M,** then the lookup table entries have **mode** = ***current operating mode***
which is the ***caption on the mode button***.

For IF width, slider table **lookup** = M.
We have to create separate lookup table entries for each operating mode.
I have separate identical LSB and USB sections in the table (*not SSB*) in order to match button captions, LSB and USB.
The **slider** has single max and min settings.
These are set to 1 and 21. Lookup entries for CW, RTTY, PSK are extended with 19, 20 21 set to 3000Hz.



Id	rig	command	code	mode	catvalue	lookup
1	FTdx101D	IF width LSB	IFWD	LSB	0	0
3	FTdx101D	IF width LSB	IFWD	LSB	1	300
4	FTdx101D	IF width LSB	IFWD	LSB	2	400
5	FTdx101D	IF width LSB	IFWD	LSB	3	600
6	FTdx101D	IF width LSB	IFWD	LSB	4	850
7	FTdx101D	IF width LSB	IFWD	LSB	5	1100
8	FTdx101D	IF width LSB	IFWD	LSB	6	1200
9	FTdx101D	IF width LSB	IFWD	LSB	7	1500

The above shows a section of lookup table in piWebCAT's editor.
I have in total:

IFWD	LSB	21 entries	CAT values 1 to 21	
IFWD	USB	21 entries	CAT values 1 to 21	
IFWD	CW	21 entries	CAT values 1 to 21	CAT value 19 to 21 added in as 3000Hz
IFWD	RTTY	21 entries	CAT values 1 to 21	CAT value 19 to 21 added in as 3000Hz
IFWD	PSK	21 entries	CAT values 1 to 21	CAT value 19 to 21 added in as 3000Hz

Any other mode uses the LSB calibration

*Setting **mode = X** makes the entries applicable to any mode.*

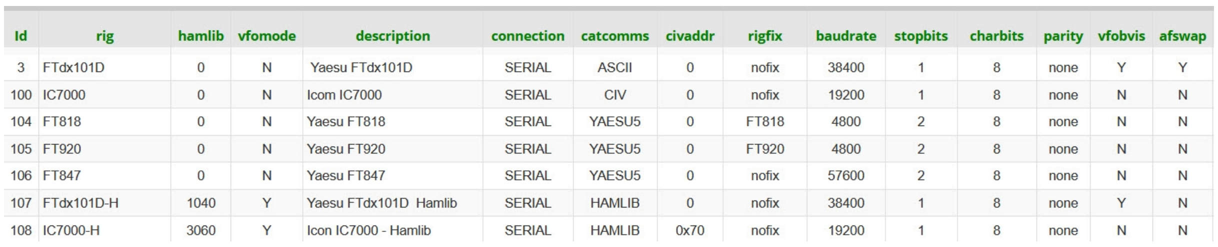
7.2 piWebCAT - Database table - rigs

The **rigs** table holds the list of radios configured in your database.

It is used:

- at startup to configure the **RPi serial port** (baudrate, stopbits, bits /character and parity)
- at startup to load **connection,** ie: SERIAL, USB or ENCAT (via EncoderCat module ... not Hamlib).
- at start up to load **catcomms**, ie; ASCII, YAESU5, CIV or HAMLIB.
  (Four of the tables are different between ASCII and CIV, eg: buttons and buttonsciv.)
- The rig fields from all the rigs table records to populate the top bar radio selector and the drop down lists when editing tables in a grid.

My database has only FTdx101D and IC7000 configured.
I put some more entries in the rigs table just for test purposes.



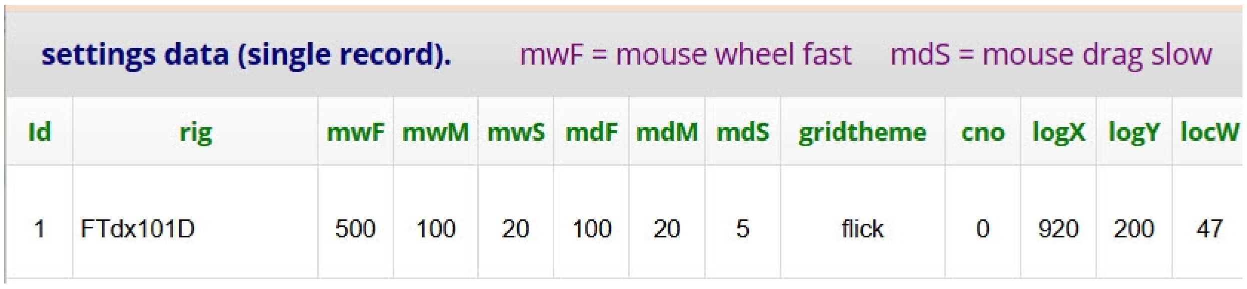
Id	rig	hamlib	vfomode	description	connection	catcomms	civaddr	rigfix	baudrate	stopbits	charbits	parity	vfobvis	afswap
3	FTdx101D	0	N	Yaesu FTdx101D	SERIAL	ASCII	0	nofix	38400	1	8	none	Y	Y
100	IC7000	0	N	Icom IC7000	SERIAL	CIV	0	nofix	19200	1	8	none	N	N
104	FT818	0	N	Yaesu FT818	SERIAL	YAESU5	0	FT818	4800	2	8	none	N	N
105	FT920	0	N	Yaesu FT920	SERIAL	YAESU5	0	FT920	4800	2	8	none	N	N
106	FT847	0	N	Yaesu FT847	SERIAL	YAESU5	0	nofix	57600	2	8	none	N	N
107	FTdx101D-H	1040	Y	Yaesu FTdx101D Hamlib	SERIAL	HAMLIB	0	nofix	38400	1	8	none	Y	N
108	IC7000-H	3060	Y	Icon IC7000 - Hamlib	SERIAL	HAMLIB	0x70	nofix	19200	1	8	none	N	N

Summary of the fields (All except **rig** and description are drop down list selectors)

- **rig** This is text. It must be spelled the same in all tables because is the identifier for a radio's records in table This is facilitated by the fact the the list of **rig** fields in the **rigs** table is offered as a drop down list in all other tables.
- **hamlib** The radio's reference number in the Hamlb database. (Only used when catcomms = HAMLIB)
- **vfomode** If Y then rigctl or rigctld starts with --vfo parameter. (Only used when catcomms = HAMLIB)
- **description** Text - no function.
- **connection** **RIG** (direct to radio from a GPIO RS232 or piWebCAT interface board) or
  ENCAT - GPIO serial connection to dedicated RPi 115200 serial port on the EncoderCAT PCB.
  USB - CAT connection is by USB from an RPi USB oscket.
- **catcomms** Which type of configuration system: ASCII, CIV, YAESU5 or HAMLIB.
- **rigfix** Invokes code functions specific to a radio or group of radios. (eg: FT818)
- **baudrate** Selector from standard baudrates.
- **stopbits** Select 1 or 2
- **charbits** Select 7 or 8
- **parity** Select none, odd or even.
- **vfobvis** Set to Y for the *Background* VFO frequency to be displayed. I set this to N (not displayed) for my
  IC7000 because there is no command to read the background VFO.
- **afswap** Set to Y to enable automatic audio swapping on VFO A/B swapping. Both receiver audio levels are stored. The background receiver audio is set to zero. See: Audio gain swapping

7.3 piWebCAT - Database table - settings

The settings table has only one record



settings data (single record). mwF = mouse wheel fast mdS = mouse drag slow

Id	rig	mwF	mwM	mwS	mdF	mdM	mdS	gridtheme	cno	logX	logY	locW
1	FTdx101D	500	100	20	100	20	5	flick	0	920	200	47

The **settings** table fields are:

• **rig**	The currently operational radio. The grid editor presents this as a drop down list of radios from the **rigs** table. It can also be set from the top bar radio selector of the three piWebCAT pages, ie: Control, Cat.config and Meter cal,
• **mwF**	Tuning - mouse wheel **fast** Hz per thumb wheel click
• **mwM**	Tuning - mouse wheel **medium** Hz per thumb wheel click
• **mwS**	Tuning - mouse wheel **slow** Hz per thumb wheel click
• **dgF**	Tuning - mouse drag **fast** Hz per dragged pixel.
• **dgM**	Tuning - mouse drag **medium** Hz per dragged pixel.
• **dgS**	Tuning - mouse drag **slow** Hz per dragged pixel.
• **gridtheme**	Editor grid theme selector. phpGrid is supplied with 26 themes. I offer the eight that are most suitbale for piWebCAT. I make no apology for their names - They are a supplied with phpGrid!!
• **cno**	The current value of the incrementing contest number, saved from the log window. If set to zero (in the log header or in this setting editor) then the log **cno** column remains blank. Saving the number in the database preserves continuity of the sequence after a power down.
• **logX**	The width of the log window in pixels.
• **logY**	The height of the log window in pixels.
• **locW**	The width of the log's loc**ator** column in pixels. Range is 0 - 140. If logW = 0 then the **locator** column in the log grid is hidden (no data is lost).

\
The table above shows my suggested settings.

7.4 piWebCAT - Database table - timing

piWebCAT's client to server requests are:

- **Timer driven repetitive tasks** (eg: frequency change, meter updates)
- **User driven tasks** such as button and slider actions.

To avoid errors due to data collisions, all client > server tasks are managed in a **queue**.
The queue has six data types in 17 slots. The slots are polled cyclically for requests.

The six data types are:

MOX	Radio Tx/Rx status. Responds to radio MOX and PTT. Read only.
METR	A meter read request (S meter or Tx meter) Read only, A and B slots.
FREQ	Frequency request. 4 slots, Read or write for VFO A or B
BUTN	Buttons - 4 slots. Read state or set state. Read or write for VFO A or B.
SLDR	Sliders - 4slots. Read state or set state. Read or write for VFO A or B.
DATA	Other CAT data requests .... not VFO specific.

The queue is scanned cyclically every 10ms (**timing** table **readqueue** field)
If a request is found, the request is processed and deleted from the queue.
- the next 10ms scan starts at the next point in the queue so that infrequent requests are not missed amongst other very frequent request types.

On fast drag or thumbwheel tuning, the next request may be generated while the previous request is still unsent in the queue slot. The new request simply overwrites the existing one.
So we lose one but we always send the latest.
The queue is designed such that we do not lose button click and slider change requests.

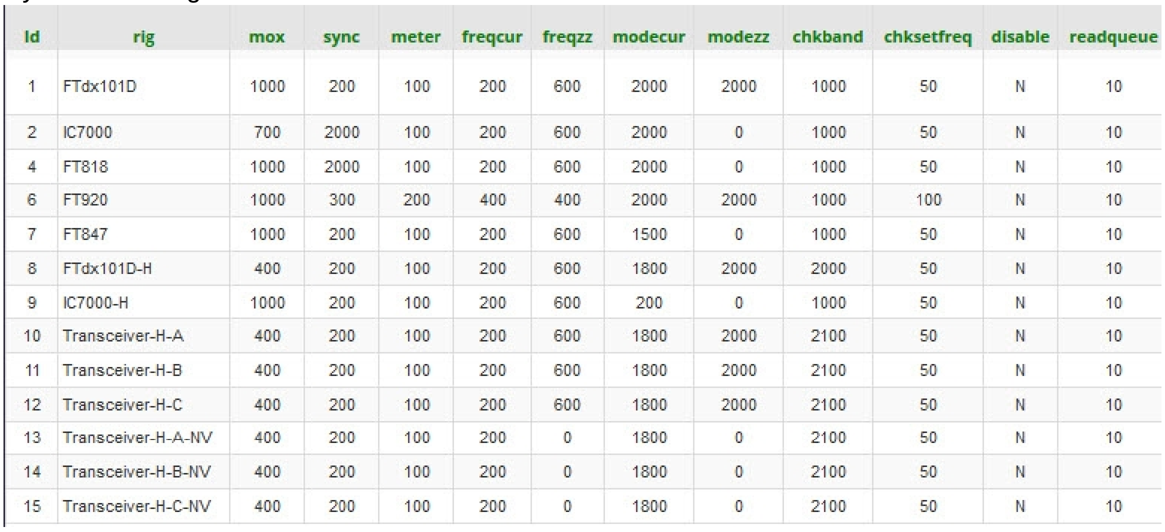
Designing for speed

For most CAT software, the overriding factor for speed limitation for repetitive tasks is the response time of the radio.
piWebCAT has the potential extra delay of every message being sent and returned via a longer pathway.
All requests arise from the client code (web browser).
The data path is:

Client code < LAN > RaspberryPi server code < serial link > radio

Because this was a potentially critical timing issue, my initial design stage was simply an S meter and tuning system to establish feasibility. There was no database use at that stage - I just used fixed coded settings for the FTdx101D. The results were promising, so the development continued.
The system was gradually refined.
The addition of queuing system was vital to avoid data collisions and to avoid losing the relatively infrequent button and slider requests.

My current timings are in the table below:



Id	rig	mox	sync	meter	freqcur	freqzz	modecur	modezz	chkband	chksetfreq	disable	readqueue
1	FTdx101D	1000	200	100	200	600	2000	2000	1000	50	N	10
2	IC7000	700	2000	100	200	600	2000	0	1000	50	N	10
4	FT818	1000	2000	100	200	600	2000	0	1000	50	N	10
6	FT920	1000	300	200	400	400	2000	2000	1000	100	N	10
7	FT847	1000	200	100	200	600	1500	0	1000	50	N	10
8	FTdx101D-H	400	200	100	200	600	1800	2000	2000	50	N	10
9	IC7000-H	1000	200	100	200	600	200	0	1000	50	N	10
10	Transceiver-H-A	400	200	100	200	600	1800	2000	2100	50	N	10
11	Transceiver-H-B	400	200	100	200	600	1800	2000	2100	50	N	10
12	Transceiver-H-C	400	200	100	200	600	1800	2000	2100	50	N	10
13	Transceiver-H-A-NV	400	200	100	200	0	1800	0	2100	50	N	10
14	Transceiver-H-B-NV	400	200	100	200	0	1800	0	2100	50	N	10
15	Transceiver-H-C-NV	400	200	100	200	0	1800	0	2100	50	N	10

Values are in milliseconds. Setting to zero stops the timer.

The timing table fields:

- **rig** The current radio - drop down selector (from **rigs** table)
  Must have same spelling throughout all the tables.
- **mox** Checking interval for radio's Rx / Tx status. 1000ms is adequate response to PTT.
- **leds** The update stepping interval for read-only LEDs buttons. See: LEDs buttons.
- **meter** S meter and Tx meter reading interval. 100ms is good. The meter needle is slightly more jerky than on the Ftdx101D but quite acceptable. I have programmed in some digital damping on meter decay.
- **freqcur** The interval for reading the radio's active VFO frequency from the radio
  This does not need to be very fast. If we are controlling with piWebCAT, it is the write to the radio interval that needs to be small to give good tuning action.
  If we are tuning on the radio, piWebCAT's response does not need to be fast.
- **freqzz** The interval for reading the radio's background frequency from the radio....can be slower.
- **modecur** Interval for checking mode change of currently selected VFO on the radio.
  Used after band and VFO A/B change. Updates appropriate buttons of Mode group and red text mode indicator to left of frequency readout.
- **modezz** Interval for checking mode change of dormant (unselected) VFO on the radio.
  Used after band and VFO A/B change.
  Updates red text mode indicator to left of frequency readout. (NOT the mode button group)
- **chkband** Interval for checking for band change on the radio.
  Band switching was a little tricky to develop. For detailed explanation see Band switching
- **chksetfreq** This interval is used when tuning the radio from piWebCAT. It needs to be fairly short to give good response. I use 50ms. As stated above, some incremental changes may be missed .. but only as a result of fast moving tuning and so this not a problem.
  The mouse drag or mousewheel tuning events update a variable: **freqSet**.
  The **chksetfreq** timer monitors freqSet and queues any change for sending to the radio.
- **disable** Drop down selector **Y** or **N**. **Y** disables all the task repetition timers except **readqueue**.
  Used for development / testing. See below:
- **readqueue** The interval between message queue polling episodes. As stated above, the queue is scanned cyclically until a message request is found. The next interval starts scanning where the last poll finished to avoid button and slider requests being missed.

Use of the disable field

Under normal piWebCAT operation, there is continuous repetitive network traffic.
ie: meter read, frequency and mode check, PTT checks. Updates for controls configured in 'sync' mode etc.

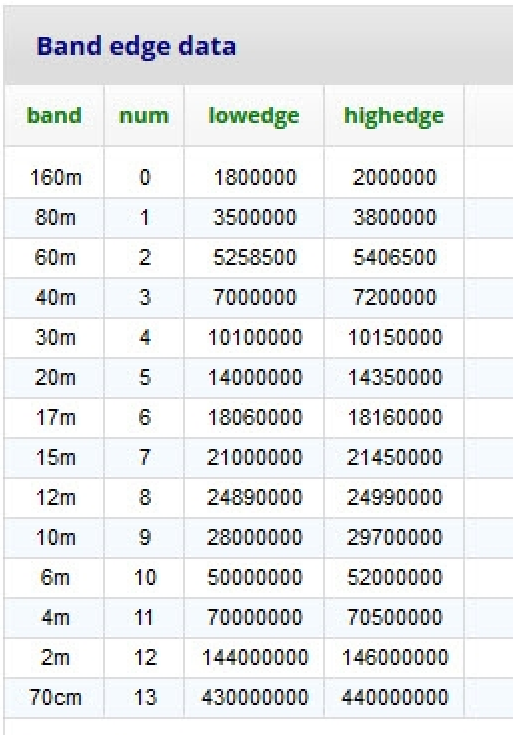
To view this traffic:
Click **R mouse** on the browser and select **Inspect element**. (Firefox developer and Chrome are good)
Selecting the **network tab** reveals a fast moving list of client-server network transactions.
You can freeze one with L mouse click and then look at **headers** and **response**.
This is hit and miss as they move too fast to make a first shot selection.

If you set **timing.disable to Y,** then all the repetitive traffic stops. You can clear the list (dustbin icon)
Then, you can test the configuration of individual buttons and sliders.
- One button click will show one transaction record which you can inspect.

7.5 piWebCAT - Table - bands

The **bands** table contains band edge frequencies for the fourteen supported bands.

Fields: band and num are read-only. They cannot be edited here.
(They can be edited with external MySQL tools such as MySQL Front.
Please do not change them - piWebCAT will not operate correctly if you do so!)



Band edge data

band	num	lowedge	highedge
160m	0	1800000	2000000
80m	1	3500000	3800000
60m	2	5258500	5406500
40m	3	7000000	7200000
30m	4	10100000	10150000
20m	5	14000000	14350000
17m	6	18060000	18160000
15m	7	21000000	21450000
12m	8	24890000	24990000
10m	9	28000000	29700000
6m	10	50000000	52000000
4m	11	70000000	70500000
2m	12	144000000	146000000
70cm	13	430000000	440000000

piWebCAT changes band in response to the radio's frequency moving ***into a new band***.

On band change, the new band's tuning scale is displayed and the appropriate band button is highlighted.

piWebCAT's band display (tuning scale etc) ***does not change on leaving the limits of the current band***.

This behaviour is determined by the band limits set in this **bands** table - which you can change.

The corresponding settings for piWebCAT's fourteen band tuning scales are separate and are fixed in the program coding.

If, for example, the limits of band were extended, then you could change the limits here.
The tuning scale would not change. Tuning into the extended region would simply go off scale.
(until someone fixed the code!).

8.1 Hamlib and piWebCAT - introduction

From the Hamlib website: https://github.com/Hamlib/Hamlib

Most recent amateur radio transceivers allow external control of their functions through a serial interface. Unfortunately, control commands are not always consistent across a manufacturer's product line and each manufacturer's product line differs greatly from its competitors.

Hamlib attempts to solve this problem by presenting a "virtual radio" to the programmer by providing an API to actions such as setting a given VFO's frequency, setting the operating mode, querying the radio of its current status and settings,and giving the application a list of a given radio's capabilities. Unfortunately, what can be accomplished by Hamlib is limited by the radios themselves and some offer very limited capability.

Hamlib and piWebCAT - connection issues.

Hamlib has been growing for 12 years and now supports 250 radios.
I am indebted to one the main current developers, Michael Black, W9MDB for his help with connecting to the rigctld API on the RPi.

piWebCAT commands all originate in the client (web browser). The commands from client to RPi server are familiar internet URL of the form:

```
http://192.168.1.117/cat/phpfiles/wcajaxdata.php?rig=FTdx101D-H&param=METR.....
```

This launches a **PHP** program file on the server (here: **wcajaxdata.php**).

For each command, the PHP code has to create and open a communication **socket** to the Hamlib rigctld API. Once the command is completed, the PHP process must **die** and so the socket is lost.
The next command has its own PHP process and must create a new socket etc etc.
This is how web servers operate and I have to live with this.

Hamlib's **rigctl** can be run at the RPi terminal command line. See Section 8.5 rigctl at the command line
This is a very convenient way of testing command syntax on your rig, eg: `\set_freq VFOA 3760000`

Programs running on the RPi communicate with **rigctl** using **rigctld** via internet protocol **socket** localhost:4532. piWebCAT's PHP code web server processes create and open a serial connection to this socket.

Hamlib does not support every available CAT command for a radio. For example, with the FTdx101D, the contour control is not supported and neither are most of the very large list of menu settings.
However, the command `\send_cmd_rx` can be used to send rig-specific commands direct to the rig.
Examples, FTdx101D-H : **contour** is supported this way. Also I have a button to toggle **SSB audio input** between front panel microphone socket and rear panel audio input (for use with Mumble VOIP)
- see section 8.14 unsupported commands

Hamlib - example - Applying FTdx101D configuration to my FT920.

CAT control of these two radios is ***very*** different. Key differences are:

- The F920 has Yaesu 5 byte command system. The FTdx101D has modern Yaesu ASCII text commands.
- The FT902 has a very limited command set.
  The FTdx101D has a large CAT command set - and I had configured a large number of these with Hamlib.

Experiment
I modified the FTdx101D-H configuration to communicate with the FT920 but did not change any commands.
..ie: I changed: USB to SERIAL, Baud rate from 38400 to 4800 and Hamlib rig number from 1040 to 1014.

Result: Slow intermittent start as Hamlib stalls on failing to process commands with no FT920 version.
Intermittently able to tune, switch VFO and change band and mode (intermiitent ... because stalls on control sync commands.) ,.... but at least we have some control, despite a very different CAT system.
The intial way forward is to disable all the controls that are not supported in FT920 CAT.

Hamlib - DUAL VFO issues

Hamlib rigctld has two modes of configuration in relation to dual VFO control:

- **--vfo mode** (where the **rigctld** start up commands includes **--vfo**.)
  This is appropriate for dual VFO / dual receiver transceivers which have separate CAT control access code for the currently selected foreground receiver AND for the background receiver.
  My FTdx101D is in this category and is used in the examples in this section.
  Note that some modern Yaesu transceivers need to use the terms Main and Sub in Hamlib's command strings, whereas many other use VFOA / VFOB. I have provided SQL scripts to automatically modify a transceiver configuration between the two options.
- **not --vfo mode**. Most Icom transceivers do not provide CAT access to the background VFO and so are probably better configured without --vfo mode. My example IC7000 configuration is in this catogory. piWebCAT can switch between the two VFOs but cannot access the background VFO.

--vfo mode is selected by setting **vfomode = Y** in the rig's record in the **radios** table.

*Note that piWebCAT only has one set of controls which connect to the current VFO. The important issue is the fact that **--vfo mode** compatible transceivers have different commands for their two VFOs. The CAT software therefore has to have a means of configuring these two command sets and for issuing the appropriate commands for the current VFO.*

I provide example configurations of --vfo mode and none --vfo mode on the SD card.

- Hamlib transceivers: **Transceiver-H-A**, **Transceiver-H-B** and **Transceiver-H-C** are a developing progression of three configurations using **--vfo mode**. (They use VFOA / VFOB. They will work with my FTdx101D but with slight VFO switching anomalies......until you change them to Main /Sub.)
- Hamlib transceivers: **Transceiver-H-A-NV**, **Transceiver-H-B-NV** and Transceiver-**H-C-NV** are a developing progression of three configurations in **none --vfo mode**. (They use VFOA / VFOB). They will operate ok with my IC7000.

Hamlib rigctl documentation https://manpages.debian.org/unstable/libhamlib-utils/rigctl.1.en.html

Hamlib example:

Setting noise reduction level in the FTdx101D and the IC7000.

The FTdx101D contains two completely separate receivers, Main and Sub.
They have their own separate NR level settings.

The IC7000 has two VFO settings but only a single receiver with a single NR level setting.

For the **FTdx101D,** the command for Main receiver is in text: **RL007;** first 0 for Main rx 07 is level (01 - 15)

For the **IC7000**, the command is in hexadecimal digits: **FE FE 70 E0 1A 05 05 01 14 07 FD**
07 is the data (00 - 15 bcd)

Using Hamlib rigctld:
For the **FTdx101D** (and many dual receiver rigs), I send: \set_level Main NR 0.500
Hamlib converts the range to 0.067 - 1.000 (piWebCAT can scale it to 1 -15 for display)

For the **IC7000** (and many single receiver rigs) I send: \set_level NR 0.500

For the FTdx101D, I use **`--vfo`** mode which handles Main and Sub receivers. I have to insert 'Main' in all commands.

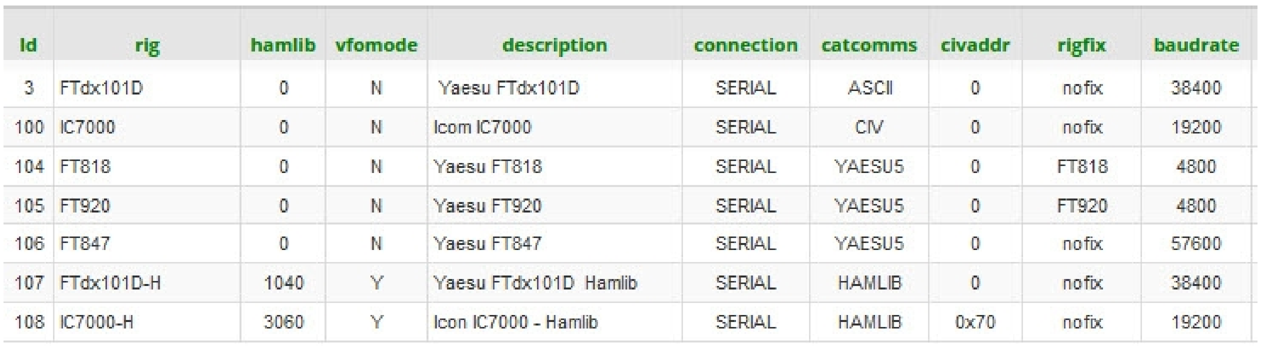
I could have used --vfo mode for the IC7000. Its command would be **`\set_level Main NR 0.500`**
...***now identical to the FTdx101D command***.

So, in summary, we can use the same command for 250 radios.
piWebCAT's data scaling and display facilities then tailor displayed data ranges to match the radio's levels.

8.2 Hamlib radio selection

Each radio has a unique identification number in Hamlib.
FTdx101D is 1040, IC 7000 is 3060. This number selects command translation to the radio's CAT commands

The piWebCAT database **rigs** table has a **hamlib** field which holds the number. (see below)
This field is only used when **catcomms** = HAMLIB.
Note that these radios appear twice in the table, eg IC7000 and IC7000-H.
(You have a free choice of name. The name becomes the link to all the radio's database records.)



Id	rig	hamlib	vfomode	description	connection	catcomms	civaddr	rigfix	baudrate
3	FTdx101D	0	N	Yaesu FTdx101D	SERIAL	ASCII	0	nofix	38400
100	IC7000	0	N	Icom IC7000	SERIAL	CIV	0	nofix	19200
104	FT818	0	N	Yaesu FT818	SERIAL	YAESU5	0	FT818	4800
105	FT920	0	N	Yaesu FT920	SERIAL	YAESU5	0	FT920	4800
106	FT847	0	N	Yaesu FT847	SERIAL	YAESU5	0	nofix	57600
107	FTdx101D-H	1040	Y	Yaesu FTdx101D Hamlib	SERIAL	HAMLIB	0	nofix	38400
108	IC7000-H	3060	Y	Icon IC7000 - Hamlib	SERIAL	HAMLIB	0x70	nofix	19200

The vfomode field.
If **vfmode = Y** then the **--vfo** startup option for **rigctl** and **rigctld** is applied.
For radios with two receivers or VFOs, it allows commands to be directed separately to the two VFOs.
See --vfo option

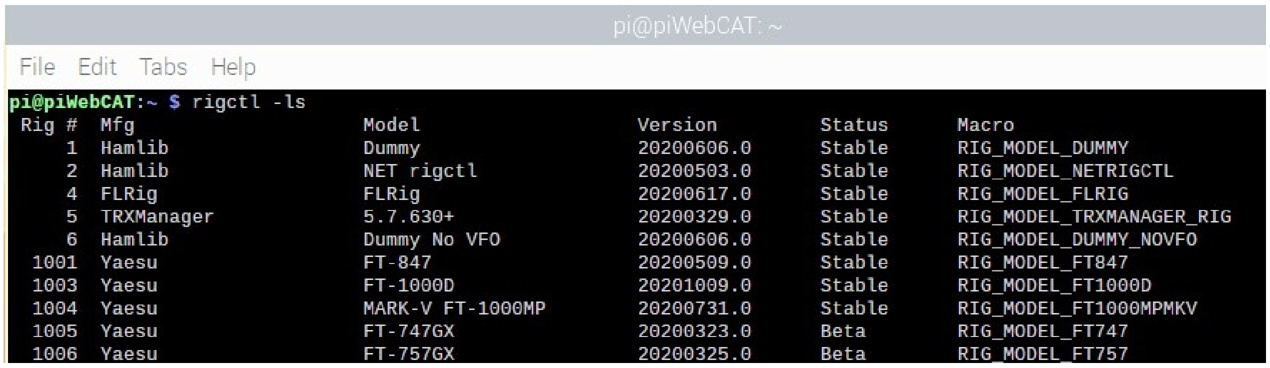
Note the **connection** field.
In the above table, FTdx101D-H uses the rig's SERIAL (RS232 connection).
However, USB connection to an RPi USB port is supported for rigs with USB connectivity.

When **catcomms = HAMLIB**, four of the database tables are Hamlib - specific:
buttonshl, catcodeshl, slidershl and **meterhl**.

Thus we have: **buttons** for ASCII and YAESU5, **buttonsciv** for Icom CIV and **buttonshl** for HAMLIB etc

The appropriate tables are automatically presented in the database editor.

The list of supported radios can be displayed using **rigctl** in the RPi terminal window.
(This image was captured by 'Snagit' software from an RPi display using VNC client on a PC.)



```
pi@piWebCAT:~ $ rigctl -ls
 Rig #  Mfg                    Model                   Version         Status      Macro
     1  Hamlib                 Dummy                   20200606.0      Stable      RIG_MODEL_DUMMY
     2  Hamlib                 NET rigctl              20200503.0      Stable      RIG_MODEL_NETRIGCTL
     4  FLRig                  FLRig                   20200617.0      Stable      RIG_MODEL_FLRIG
     5  TRXManager             5.7.630+                20200329.0      Stable      RIG_MODEL_TRXMANAGER_RIG
     6  Hamlib                 Dummy No VFO            20200606.0      Stable      RIG_MODEL_DUMMY_NOVFO
  1001  Yaesu                  FT-847                  20200509.0      Stable      RIG_MODEL_FT847
  1003  Yaesu                  FT-1000D                20201009.0      Stable      RIG_MODEL_FT1000D
  1004  Yaesu                  MARK-V FT-1000MP        20200731.0      Stable      RIG_MODEL_FT1000MPMKV
  1005  Yaesu                  FT-747GX                20200323.0      Beta        RIG_MODEL_FT747
  1006  Yaesu                  FT-757GX                20200325.0      Beta        RIG_MODEL_FT757
```

8.3 Hamlib - Installation / update on the Raspberry Pi

Hamlib is distributed ready installed on the piWebCAT SD card. This is unlikely to be the latest update.

At times of active development, updates may occur daily and may be related to very recent modern radios.

You may wish to perform an initial update, particularly if you have very a modern radio.

Update procedure.

Start the RPi and access its screen with VNC client (simpler than plugging in mouse, keyboard and monitor)

Start **File Manager**. This will open in **/home/pi.** Enter the **Downloads** subfolder.

If present, delete Hamlib-master.zip and the Hamlib-master folder.

Internet: Use the web browser (Chromium) to navigate to https://github.com/Hamlib/Hamlib

Click the green **Code** download button. Click **Download ZIP**.
Hamlib-master.zip will download to /home/pi/Downloads

Using **File Manager**, in folder **/home/pi/Downloads** double-click **Hamlib-Master.zip**.
Click the **Extract files** button. Change the destination from /tmp to **/home/pi/Downloads**.

Click the **Extract** button. Wait a few seconds. Folder /home/pi/downloads/Hamlib-master is created.

Close FIle Manager.

Open RPi **terminal** and enter following sequence of commands. (It should open in /home/pi)

```
$  cd Downloads/Hamlib-master

$  chmod 744 bootstrap

$   ./bootstrap

$  ./configure --prefix=/usr/local --enable-static

$  make

$  sudo make install

$ sudo ldconfig
```

8.4 Hamlib - rigctl, rigctld documentation

This website/document gives examples of Hamlib's commands.
However, no attempt is made to republish the list of option settings and commands.

The documentation is at: https://manpages.debian.org/unstable/libhamlib-utils/rigctl.1.en.html

From the **Hamlib utilities** section, Download and print: **rigctl** and **rigctld** (pdf documents)

rigctl is used to manually send commands from the Linux command line (ie: in RPI terminal)
It is invaluable in testing command strings on the radio.

rigctld provides a TCP serial socket through which piWebCAT sends the commands to the radio.

Both of the documents have sections: OPTIONS and COMMANDS.

OPTIONS are startup settings for **rigctl** or **rigctld.**
eg: -m (radio number) -s (baud rate) -r (device, eg:/dev/ttyUSB0)

COMMANDS are the actual commands sent to the **rigctl** once **rigctl** or **rigctld** are running.

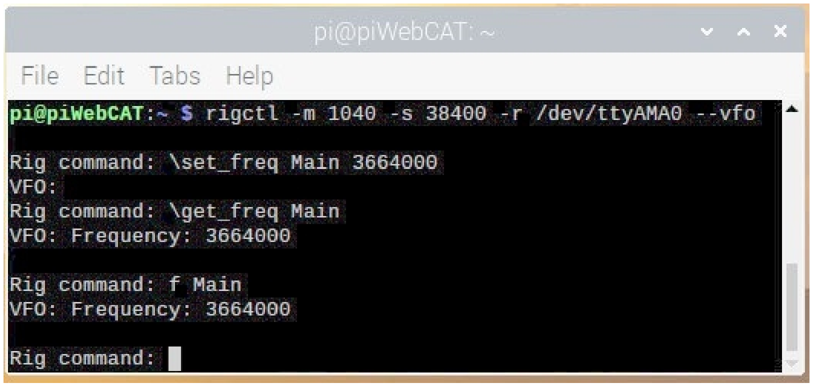
The OPTIONS sections are slightly different between **rigctl** and **rigctld.**
In the COMMANDS section, the actual commands are essentially the same.

The --vfo option

- **--vfo** is a startup option for **rigctl** and **rigctld** for radios with two receivers or VFOs.
- It allows commands to be directed separately to the two VFOs.
- Some radios use the labels VFOA and VFOB. Some use Main and Sub.
- The position of 'Main' etc in the command is illustrated here by: **\set_level Main NR 0.500**
  (Where NR mean noise reduction level)
- **--vfo mode** is selected by setting **vfomode = Y** in the radio's entry in the **rigs** table

*If **--vfo** is used, then ALL commands must have a **Main** or **Sub** etc parameter, even if they don't use it.*
For example: my RF power level control for the FTdx101D is:
\set_level Main RFPOWER 0.500
*- Here **Main** is just a **dummy** parameter. RFPOWER is not VFO -specific.*
*- I could have used **\set_level Sub RFPOWER 0.5000***

8.5 Hamlib - Using rigctl at the command line



```
pi@piWebCAT:~ $ rigctl -m 1040 -s 38400 -r /dev/ttyAMA0 --vfo

Rig command: \set_freq Main 3664000
VFO:
Rig command: \get_freq Main
VFO: Frequency: 3664000

Rig command: f Main
VFO: Frequency: 3664000

Rig command:
```

In the above example, to start **rigctl**, I entered: **`rigctl -m 1040 -s 38400 -r /dev/ttyAMA0 --vfo`**
`-m 1040` selects the FTdx101D
`-s 38400 sets the baudrate`
`-r /dev/ttyAMA0` selects the RPi serial port connected to the radio's RS232 connector.
(For USB, I would have entered **`-r /dev/ttyUSB0`**)
`--vfo` selects **vfo mode** (see --vfo option)

I then entered three commands:

• **`\set\freq Main 3664000`**	Sets the main VFO frequency
• **`\get_freq Main`**	Read the main VFO frequency
• **`f Main`**	Read the main VFO frequency (short command format)

These commands are ***rigctl*** commands. **rigctl** translates them into the radio's CAT commands.

Using rigctl in configuring a radio

rigctl at the command line allows:

- Testing of commands on the radio
- Examination of responses - eg: to check ranges of returned data.
- List of available command options for the radio
- Testing the use of \send_command_rx for commands not supported by Hamlib.

Checking ranges

Example for FTdx101D
The CAT manual says that **RF gain** is sent and received as R G P1 P2 P2 P2 ;
where P1 = 0 / 1 for Main /Sub and P2 is 000 to 255

Using the Main receiver:
We set RF gain at minimum and send rigctl command **\get_level Main RF** This returns **0.000**.
We set RF gain at maximum and send rigctl command **\get_level Main RF** This returns **1.000**
So the range returned to piWebCAT is **0.000 to 1.000**.

In the **slidershl** table, the **min** and **max** fields control the slider range.
We set **min = 0.000** and **max = 1.000**.
piWebCAT will internally automatically scale these returned CAT levels to the working slider range of 0 - 400.
(All sliders have internal HTML range 0 - 400)
Conversely, on user slider action, the slider position 0 - 400 will be sent as 0.000 - 1.000

piWebCAT also displays a **numeric value** against the slider.
You have a choice. You can configure to display 0.00 - 1.00 0 - 100 0 - 255 or whatever you choose.
The **mult**, **divide** and and **decpoint** fields convert the CAT value (0.000 - 1.000) to the displayed value.

Setting **mult = 255** and **divide = 1** would display 0.000 - 1.000 as 0 - 255 (the actual rig CAT range)
Setting **mult = 100** and **divide = 1** and **units = "%"** would display 0 - 100%

rigctld at the command line

You can also test with **rigctld** at the command line,
eg, type: **`rigctld -m 1040 -s 38400 -r /dev/ttyAMA0 --vfo`**
Then open another terminal window on the RPi and use Telnet:
Enter: **`telnet 127.0.0.1 4532`** This will connect telnet to the rigcrld socket on port 4532.

8.6 Hamlib - rigctld <> piWebCAT interface

From the downloaded Hamlib document - RIGCTLD(8)

***rigctld** communicates to a client through a TCP socket using text commands shared with **rigctl**. The protocol is simple; commands are sent to **rigctld** on one line and **rigctld** responds to "get" commands with the requested values, one per line, when successful, otherwise, it responds with one line "RPRT x", where x is a negative number indicating the error code. Commands that do not return values respond with the line "RPRT x", where x is zero when successful, otherwise is a negative number indicating the error code. Each line is terminated with a newline '\n' character.*

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite.

rigctld opens a serial TCP socket. Sockets are essentially LAN / internet ports which handle two-way serial data.
The rigctld socket is on **IP address 127.0.0.1** (localhost) with **port number 4532**.
127.0.0.1 is an internal IP address which has access restricted to other processes within the the device (ie: RPi)

piWebCAT's Apache web server uses PHP commands to communicate with socket:
These are: socket_create(..) socket_connect(..) socket_read(..) socket_write(..) and socket_close(..)

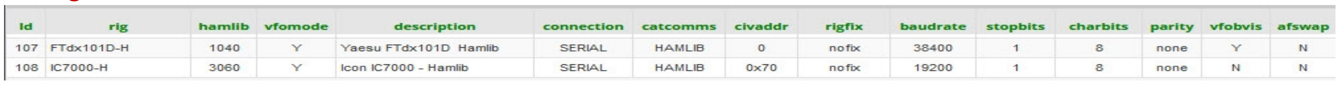
The data pathway is:
browser javascript < ajax > server PHP < socket > rigctld (translate to rig CAT) < serial or USB > rig

rigctld has to be started at the Linux command line.
This done at piWebCAT startup by a **shell_exec(command string**) command which allows PHP code to issue a command string as if it were typed into the RPI terminal at the command line.
A typical command string is:

```
shell_exec("rigcltd -m 1040 -r /dev/ttyUSB0 -s 38400 --vfo
                         --set-conf=data-bits=8stopbits=1 -c 00"  > /dev/null &);
```

1040 (rig no), **/dev/ttyUSB0** (device), **38400** (baudrate), **8** (databits) and **1** (stopbits) are all read from the radio's user-entered configuration settings in the database **rigs** table as shown below.
-c 00 is the Icom CI-V address which is only used for Icom radios.
> /dev/null & allows rigctld to continue running while PHP moves on to the next command.

You do not have to deal with this command string. It is generated automatically from the parameters in the rigs table, as shown below:



Id	rig	hamlib	vfomode	description	connection	catcomms	civaddr	rigfix	baudrate	stopbits	charbits	parity	vfobvis	afswap
107	FTdx101D-H	1040	Y	Yaesu FTdx101D Hamlib	SERIAL	HAMLIB	0	nofix	38400	1	8	none	Y	N
108	IC7000-H	3060	Y	Icon IC7000 - Hamlib	SERIAL	HAMLIB	0x70	nofix	19200	1	8	none	N	N

Closing piWebCAT with rigctld

piWebCAT will close down rigctld if current **catcomms** is HAMLIB and we are in the control window, as follows:

- **Exit** button clicked.
- On changing rig.
- On switching to the **config** or **metercal** windows.

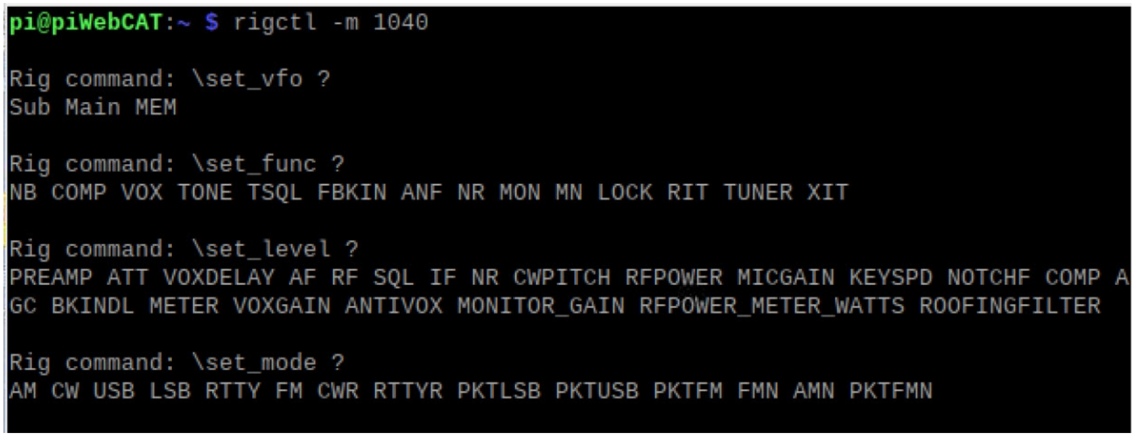
This is done because rigctld startup cannot apply changed startup parameters if rigctld is already running.
(eg: a change of baudrate)
.
You can also stop rigctld at the RPi command line using: `sudo killall rigctld`

8.7 Hamlib - rigctl / rigctld - displaying controls supported for your rig

For a full command list, see Hamlib documentation. https://github.com/Hamlib/Hamlib/wiki/Documentation

Before embarking on configuring piWebCAT / Hamlib for your rig, you need to list the available control functions. This is done using **rigctl** at the RPi terminal command line as shown below.
I entered **`rigctl -m 1040`** (only the -m parameter is needed here - It selects the rig = FTdx101D = 1040)

All commands have a short form. I could have used **`U ?`** instead of **`\set_func ?`** .
I have used the long form in piWebCAT example configurations to make the commands easier to understand.



```
pi@piWebCAT:~ $ rigctl -m 1040

Rig command: \set_vfo ?
Sub Main MEM

Rig command: \set_func ?
NB COMP VOX TONE TSQL FBKIN ANF NR MON MN LOCK RIT TUNER XIT

Rig command: \set_level ?
PREAMP ATT VOXDELAY AF RF SQL IF NR CWPITCH RFPOWER MICGAIN KEYSPD NOTCHF COMP A
GC BKINDL METER VOXGAIN ANTIVOX MONITOR_GAIN RFPOWER_METER_WATTS ROOFINGFILTER

Rig command: \set_mode ?
AM CW USB LSB RTTY FM CWR RTTYR PKTLSB PKTUSB PKTFM FMN AMN PKTFMN
```

VFOA / VFOB or Main /Sub --vfo mode *** IMPORTANT ****

See also section 2.10 Icom - --vfo mode

--vfo mode is selected at the RPI terminal command line by adding **--vfo** to the end of **rigctl** start text:
eg: **`rigctl -m 1040 -s 38400 -r /dev/ttyUSB0 --vfo`**
where `1040 = code for FTdx101D, 38400 is baudrate
and /dev/ttyUSB0 is the connection for USB (serial connection is /dev/ttyAMA0)
--vfo mode in your piWebCAT configuration is selected by setting **vfomode=Y** in the **radios** table.

if you are using **--vfo mode**, then all commands must contain a VFO parameter.
This requirement applies even if the command is not VFO - specific.
eg **`\get_level VFOA NR`** (get noise reductions level for VFOA receiver)
and **`\get_level VFOA RFPOWER`** - this is not VFO dependant but it needs VFOA as dummy parameter.

Most **Icom** transceivers and several others do not have CAT access to the background receiver (B or Sub)
All commands act only on the active receiver (A or Main).
For these transceivers I suggest not using --vfo mode.
So the commands are **`\get_level RFPOWER`** etc (no VFO parameter needed)
The only command that needs the VFO identity is the command to switch VFOs. eg: **`\set_vfo VFOA`**

In the above screen dump, I use **`\set_vfo ?`** to find out how Hamlib labels the VFOs for my transceiver
- The return is **Sub Main**. This was for the FTdx10D.

The FTdx101D has two completely separate receivers with separate settings.
All control commands (eg: noise reduction, gain, bandwidth etc) can be addressed to either receiver, whether or not it is the current foreground receiver or the background receiver. *So we use --vfo mode.*

My evolving progression of Transceivers-H-A toC uses --vfo mode. (see Learning section 8).

I discovered that the FTdx101D would accept VFOA/VFOB as alternative to Main/Sub .. with two exceptions.
The exceptions are \set_vfo and get_vfo and \set_mode and \get_mode.
I therefore configured all the Hamlib --vfo example transceivers in this way.
The reason for doing this was so that you would not have to edit numerous instances of Main to VFOA etc.

\set_func (U) \get_func (u)

The commands set and read on / off switches. (Switches with more than two states tend to use \set_level)

In piWebCat, they will usually be assigned to buttons with **action = T** (on/off toggling).
We set 1 for ON and 0 for OFF.

Examples:

\set_func Main NR 1 **\get_func Main NR** Set and read noise reduction on / off status

\set_func VFOA VOX 0 **\get_func VFOA VOX** Set and read VOX on/ off status

In piWebCAT configuration, we use **\set_func Main NR #** in table **catcodeshl**.
The # is substituted by the value (**0** or **1**) assigned in table **buttonshl**. This is discussed below in more detail.

\set_level (L) \get_level (l)

The commands set and read controls with continuous level or multi-value adjustment.
In piWebCAT, these are used with:

- slider controls
- button groups with more than two states

Examples:
\set_level Main NR 0.500 **\get_level Main NR** Set and read noise reduction level.
\set_level VFOA VOXGAIN 0.400 **\get_func VFOA VOXGAIN** Set and read VOX gain.

VOXGAIN is not VFO-specific and so VFOA above is a dummy value. See The --vfo option

Note that **rigctl** and **riigctld** use standard ranges which are often 0.000 to 1.000.
The ranges used can be checked at the rigctl command line.
See Checking ranges in rigctl at the command line
Note that piWebCAT's configuration provides for any range to be scaled to the span of a slider control and for your choice of text displayed value.

In piWebCAT configuration, we use **\set_level Main NR #** in table slidershl.
The # is substituted by an appropriate value scaled from the slider position.
This is discussed below in more detail
.

\set_vfo (V) \get_vfo (v)

The commands set and read current VFO selection

Examples:
\set_vfo Main **\set_vfo Sub** **\set_vfo VFOA** **\get_vfo**

These options will be linked to the VFOA and VFOB buttons.
These buttons have a fixed **code = VFO** which must not be changed.
The fixed **VFO** code links them internally to the **Swap VFO** button and also connects them to piWebCAT's audio gain swapping system. See: Audio gain swapping

In the piWebCAT configuration, we use **\set_vfo #** in table **catcodeshl**.
The # is substituted by the value (Main, Sub, VFOA etc) assigned in table **buttonshl**.
This is discussed below in more detail.

8.8 Hamlib - rigctld - configuring buttons

The following section deals with configuring buttons for those commands that are supported by **rigctld**.
(Unsupported commands can use **\send_cmd_rx** which is discussed separately)
rigctld commands are text based and so configuration is similar to the ASCII configuration system.
Prior reading of the section on Database tables - configuration is assumed.
Some reading of ASCII text CAT - configuration would be useful.

Tables buttonshl and catcodeshl

As with the other configuration systems (buttons, buttonsciv)

- Table **buttonshl** contains button data extracted at startup from the server database to a client data array. The data includes: caption, colour, active status, group/toggle/single status and data values to be passed to the server or interpreted from the server. Also the **code** and **vx** values to link to the server.
- Table **catcodeshl** contains the **rigctld** command strings (**readmask** and **setmask** fields). It also contains client links: **code** and **abx**.

Most buttons definitions in **buttonshl** link to server commands in **catcodehl**.
Exceptions are (Using example from my FTdx101D configuration:

- **More** (code fixed = **MORE**) Launches the popup window with 24 extra buttons.
- **MPad** (code fixed = **MPAD**) Launches the memory channel selector keypad.
- **rep 10**, **rep 20** (**code** fixed = **CWRP**) CW message repeat is an interval timer in client javascript code.
- **repeat** (code fixed = **RPGO**) CW message repeat on / off - built in client function.
- **Tx metering option buttons** - these link to the **meter** table records.
- **Slider set-to-default buttons** (buttons 51 to 56, 89 - 91)

Table buttonshl fields

• **rig**	The current radio - selector (from rigs table). Must have same spelling through the tables.
• **description**	Descriptive text- no function
• **colour**	Button's background colour at startup. Can be a standard HTML color, eg teal,indigo etc or a numeric colour: ie: #RRGGBB eg #223344 where 0x22 is red level (0x00 - 0xFF)
• **caption**	The caption to be applied to the button at start up. Must fit the button's width. Try something and then observe. (Lower case letters are MUCH narrower!)
• **btnno**	The button's unique, fixed, numeric identifier. See: Button and slider numbering
• **active**	Y = button active N = button inactive S = active + sync (repetitive state update from rig) **L** = sync and 'LED' read-only indicator lamps (See: LED buttons)
• **code**	3 or 4 upper case characters. This links a button command to its action on the server. It must match the **code** for the linked record in **catcodes**.
• **vx**	**V** for client to send A or B according to current VFO. (A and B catcodes records) **X** to send X (single non-VFO dependent server record. **U** no action to server. (U = do not participate in buttons state store and retrieve) See: vx and abx
• **getset**	(Hamlib only) **--**, **G**, **S** or **GS**. **G** for get function. **S** for set function.
• **action**	**S** = single momentary. Button flashes briefly. Sends **von** field value. **T** = toggled. Alternates on/off. Client code highlights it when on and remembers its current state. Sends **von** or **voff** column value. **G** = grouped. Is in a group of other **G** buttons *with the same **code**.* Only one of the group is highlighed. The data value associated with each button is in the **nset** field (Corresponding **nans** field is used to match data read from the rig). **M** = a meter button. **R** = a slider reset buttons
• **seton**	ON value for an on/off (toggling) button ie: action = **S** or **T** (usually 1)
• **setoff**	OFF value for an on/off button ie: action = **S** or **T** (usually 0)
• **anson**	ON value to match a button state command received from the radio.
• **ansoff**	OFF value to match a button state command received from the radio.
• **nset**	The send value for a button that is in a **button group**
• **nans**	The answer matching value for a button in a **group** (not always the same as sent)

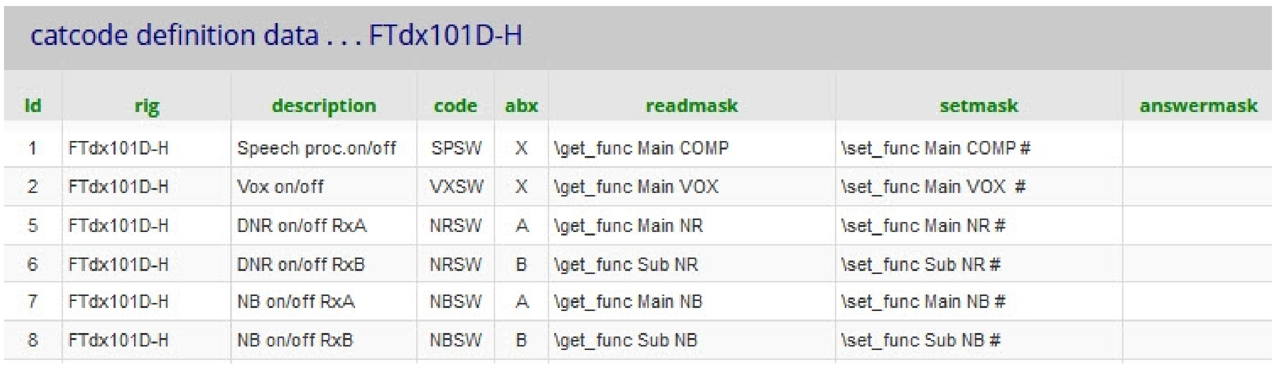
8.9 Hamlib -- rigctld - table catcodeshl

The following section deals with configuring buttons for those commands that are supported by **rigctld**.
(Unsupported commands can use **\send_cmd_rx** which is discussed separately)
rigctld commands are text based and so configuration is similar to the ASCII configuration system.
Prior reading of the section on Database tables - configuration is assumed.
Some reading of ASCII text CAT - configuration would be useful.

catcodeshl is used on the server to communicate with the radio for buttons and frequency.
(None of its data is loaded to the client)

- **rig** The current radio - drop down selector (from **radios** table)
  Must have same spelling through the tables.
- **description** Descriptive text- no function
- **code** The link to the buttons table - your choice unless has fixed function coded in piWebCAT
  *Note that for frequency reading and setting, the code must be **FREQ*** (hard coded)
- **abx** **A** or **B** if there is pair of entries one for each VFO (eg: Mute RxA, RxB)
  X if A or B not relevant. (eg:swap VFOs, Tuner etc) See also: vx and abx
- **readmask** The character string used to read from the rig via rigctld.
- **setmask** The character string used to write to the rig via rigctld.
  Contains # which is substituted on button action by the data from buttonshl.
- **answermask** *This is only used for commands not supported by rigctld where we use \send_cmd_rx.*
  It is the character pattern of the answer and has the same format as in the ASCII system.
  See Command masks

Note that **catcodeshl** may contain either one record **(abx = X**) or two records (**abx** = **A** or **B**) for each **buttonshl** button or group of buttons. This the same in the ASCII, YAESU5 and CIV configuration systems.
See: README - vx and abx



catcode definition data . . . FTdx101D-H

Id	rig	description	code	abx	readmask	setmask	answermask
1	FTdx101D-H	Speech proc.on/off	SPSW	X	\get_func Main COMP	\set_func Main COMP #	
2	FTdx101D-H	Vox on/off	VXSW	X	\get_func Main VOX	\set_func Main VOX #	
5	FTdx101D-H	DNR on/off RxA	NRSW	A	\get_func Main NR	\set_func Main NR #	
6	FTdx101D-H	DNR on/off RxB	NRSW	B	\get_func Sub NR	\set_func Sub NR #	
7	FTdx101D-H	NB on/off RxA	NBSW	A	\get_func Main NB	\set_func Main NB #	
8	FTdx101D-H	NB on/off RxB	NBSW	B	\get_func Sub NB	\set_func Sub NB #	

In the above table:

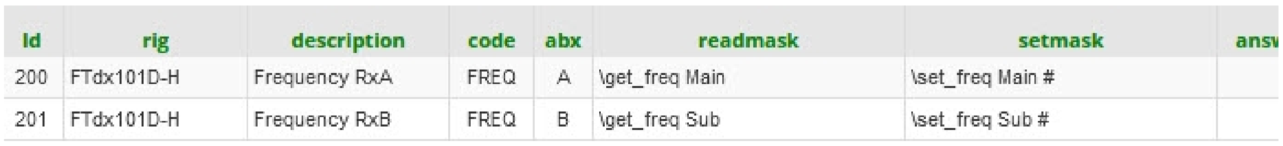
code = SPSW is speech processor on / off

readmask is the command sent to **rigctld**. ie: **`\get_func Main COMP`**
This command is not VF0/receiver-specific, but FTdx101D needs **--vfo** mode and so the word **Main** must be present (as a dummy) in the command string.
The response is a numeric and is handled internally. The **answermask** field is not used

setmask is **\set_func Main COMP #.**
The # is substituted by the **seton** or **setoff** fields value sent from the **buttonshl** record in the client.
(or from the nset field fro grouped buttons)
 - so we actually sent to **rigctld**: **`\set_func Main COMP 0`** or **`\set_func Main COMP 1`**

Note that the Vox and Speech proc. switching have a single record with **abx = X**.
and that DNR and NB each have two records (for Main rx and Sub rx) with **abx = A** and **abx = B**
See: vx and abx

Frequency
Note that **catcodeshl** contains the records for reading and setting VFO frequency.
These must use **code=FREQ** and **abx** = **A** and **B**.
They are linked internally to piWebCAT's tuner.



Id	rig	description	code	abx	readmask	setmask	answ
200	FTdx101D-H	Frequency RxA	FREQ	A	\get_freq Main	\set_freq Main #	
201	FTdx101D-H	Frequency RxB	FREQ	B	\get_freq Sub	\set_freq Sub #	

8.10 Hamlib -- rigctld - table slidershl

The following section deals with configuring sliders for those commands that are supported by **rigctld**.
(Unsupported commands can use **\send_cmd_rx** which is discussed separately)
rigctld commands are text based and so configuration is similar to the ASCII configuration system.
Prior reading of the section on Database tables - configuration is assumed.

Table slidershl

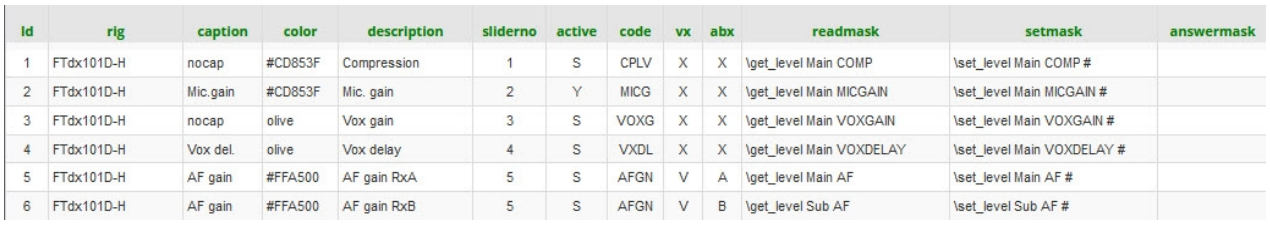
As with the other configuration systems **slidershl** contains both client and server configuration data.
(Whereas, for buttons, its separate in **buttonshl** (client) and **catcodeshl** (server)

- Client **slidersshl** data is extracted at startup from server database to client data array.
  The data includes: **caption**, **color**, **sliderno**, **active** status and **code** and **vx** for linking to the server.
  It also contains **min** and **max** to match slider position to CAT data range and **mult**, **divide**, **decpoint** and **units** to scale and present the numeric display to the right of the slider.
- Server **catcodeshl** contains the **rigctld** command strings (**readmask** and **setmask** fields).
  It also contains client links **code** and **abx**.

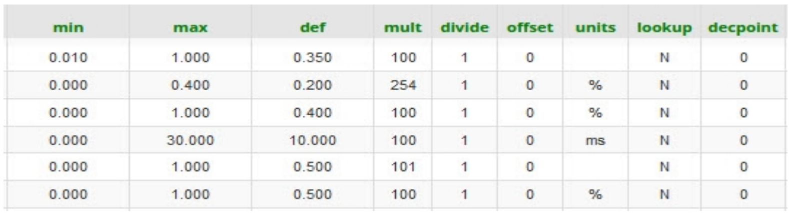
sliderhl fields:

- **rig** The current radio - drop down selector (from **radios** table). Needs same spelling in all tables.
- **description** Descriptive text- no function
- **caption** The caption to the left of the slider. Only applies to the central column of sliders with no adjacent on/off button with identifying caption.
- **sliderno** The sliders's unique, fixed, numeric identifier. See: Button and slider numbering
- **active** Y = slider active N = slider inactive (drop list selector)
- **code** 3 or 4 upper case characters. This links to a slider movement action on the client to its action processes on the server. It is transmitted to the server with data (jobdata)
  See below for more explanation.
- **vx** **vx = V, X or U**. Client field that controls storage of latest setting for each VFO.
  vx is set to **V** for dual receiver settings for the radios stores a different value per receiver.
- **abx** **A** or **B** if there is pair of entries one for each VFO eg: DNR level, RF gain
  X if A or B not relevant. (eg: RF power level, mic gain etc) See: vx and abx
- **readmask** The character string used to read from the rig via rigctld.
- **setmask** The character string used to write to the rig via rigctld.
  Contains **#** which is substituted on slider movement by a value scaled from slider position.
- **answermask** *This is only used for commands not supported by rigctld where we use \send_cmd_rx.*
  It is the character pattern of the answer and has the same format as in the ASCII system.
  See Command masks
- **min** The minimum of the CAT data (ie: before scaling etc)
- **max** The maximum value of the CAT data.
  min and **max** ensure that the range of the sent CAT value spans the limits of the slider and that the slider response to radio initiated change also fits the range,
- **def** Default value - set by the associated reset button (not available on all sliders)
- **mult** Multiplier to scale CAT value for display.
- **divide** Divider to scale CAT value for display.
  eg: use (CAT * 100) / 255 to scale 0-255 to 0 - 100
  (Multiply first then divide).
- **offset** Offset applied to display value. eg: offset = -50 scales 0 -100 to -50 to +50.
- **units** Units after displayed value eg: Hz kHz etc
- **lookup** If lookup = **Y** then displayed value is taken from lookup table entries with matching code.
  Lookup uses the CAT value after scaling (if any).
  If lookup = **M** then displayed value from lookup table entry with matching **code** and lookup table **mode** field = current operating mode (eg: USB, LSB etc).
  For lookup = **M**, lookup table mode must be the mode button's caption -
  ie: LSB, USB, CW etc (NOT SSB) See Lookup table .
- **decpoint** Add decimal point = dec places. eg: **decpoint** = 3 for 3000Hz to 3.000kHz.

Table slidershl sample:



Id	rig	caption	color	description	sliderno	active	code	vx	abx	readmask	setmask	answermask
1	FTdx101D-H	nocap	#CD853F	Compression	1	S	CPLV	X	X	\get_level Main COMP	\set_level Main COMP #	
2	FTdx101D-H	Mic.gain	#CD853F	Mic. gain	2	Y	MICG	X	X	\get_level Main MICGAIN	\set_level Main MICGAIN #	
3	FTdx101D-H	nocap	olive	Vox gain	3	S	VOXG	X	X	\get_level Main VOXGAIN	\set_level Main VOXGAIN #	
4	FTdx101D-H	Vox del.	olive	Vox delay	4	S	VXDL	X	X	\get_level Main VOXDELAY	\set_level Main VOXDELAY #	
5	FTdx101D-H	AF gain	#FFA500	AF gain RxA	5	S	AFGN	V	A	\get_level Main AF	\set_level Main AF #	
6	FTdx101D-H	AF gain	#FFA500	AF gain RxB	5	S	AFGN	V	B	\get_level Sub AF	\set_level Sub AF #	



min	max	def	mult	divide	offset	units	lookup	decpoint
0.010	1.000	0.350	100	1	0		N	0
0.000	0.400	0.200	254	1	0	%	N	0
0.000	1.000	0.400	100	1	0	%	N	0
0.000	30.000	10.000	100	1	0	ms	N	0
0.000	1.000	0.500	101	1	0		N	0
0.000	1.000	0.500	100	1	0	%	N	0

Here are 6 records of the **slidershl** table (split to fit)
The first four records are singe items which are not vfo/receiver-specific and so have: **vx = X abx=x**

Id 5 and 6 are AF gain, one Main rx, one Sub rx.

Client config: **vx = V** so client sends current vfo, A or B.

Server response: **abx = A** (Main rx)
abx = B (Sub rx)

Using compression level as an example from the above table:

readmask is the command sent to **rigctld**. ie: **`\get_level Main COMP`**
This command is not VF0/receiver-specific, but piWebCAT uses **--vfo** mode and so the word **Main** must be present (as a dummy) in the command string.

The response is a numeric and is handled internally. Experimenting with **rigctl** revealed a range of 0.010 to 1.000
The rig range is observed to be 1 - 100 (CAT manual says 0 - 100)
See: rigctl at the command line, checking ranges.

We set min = 0.010 and max = 1.000. piWebCAT then scales so that the data range gives the full slider travel.
We set mult = 100 and divide =1. This is purely for the displayed value which will then be 1 to 100.

The **answermask** field is not used. It is only used with **\send_cmd_rx** for unsupported CAT commands.

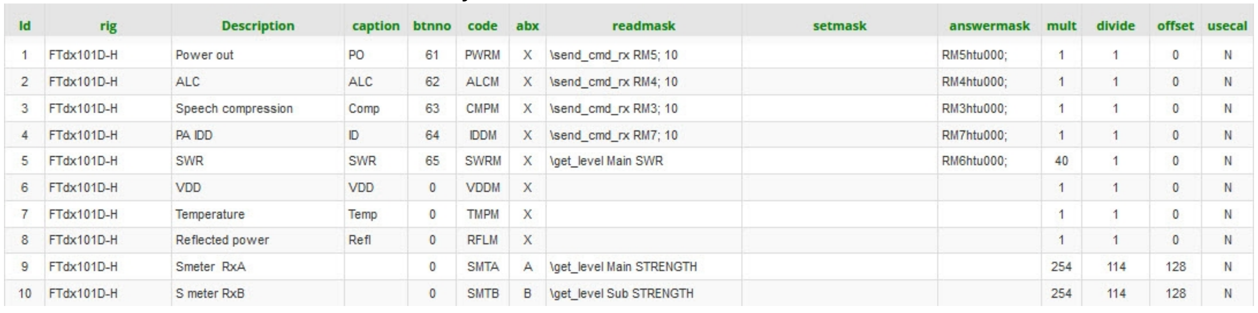
setmask is **\set_level Main COMP #.**
The # is substituted by the data value which is derived from slider position.
Again, the **min** and **max** settings perform a similar scaling in reverse giving a CAT value range of 0.010 to 1.000.
- so we actually sent to **rigctld**: **`\set_level Main COMP 0.450 etc`**

8.11 Hamlib -- rigctld - table meterhl

piWebCAT has an S meter on receive and five button-selected meter options on transmit.

The five Tx buttons are set up in the **buttonshl** table.
The **meterhl** table controls the subsequent repetitive meter reading.
The **timing** table sets the meter repetition interval
The link between the **buttonshl** table and the **meterhl** table is the **btnno** field values, 61 to 65.

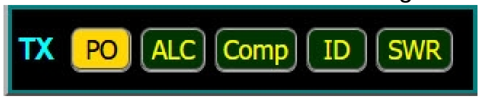
The meterhl table is shown below for my FTdx101D



Id	rig	Description	caption	btnno	code	abx	readmask	setmask	answermask	mult	divide	offset	usecal
1	FTdx101D-H	Power out	PO	61	PWRM	X	\send_cmd_rx RM5; 10		RM5htu000;	1	1	0	N
2	FTdx101D-H	ALC	ALC	62	ALCM	X	\send_cmd_rx RM4; 10		RM4htu000;	1	1	0	N
3	FTdx101D-H	Speech compression	Comp	63	CMPM	X	\send_cmd_rx RM3; 10		RM3htu000;	1	1	0	N
4	FTdx101D-H	PA IDD	ID	64	IDDM	X	\send_cmd_rx RM7; 10		RM7htu000;	1	1	0	N
5	FTdx101D-H	SWR	SWR	65	SWRM	X	\get_level Main SWR		RM6htu000;	40	1	0	N
6	FTdx101D-H	VDD	VDD	0	VDDM	X				1	1	0	N
7	FTdx101D-H	Temperature	Temp	0	TMPM	X				1	1	0	N
8	FTdx101D-H	Reflected power	Refl	0	RFLM	X				1	1	0	N
9	FTdx101D-H	Smeter RxA		0	SMTA	A	\get_level Main STRENGTH			254	114	128	N
10	FTdx101D-H	S meter RxB		0	SMTB	B	\get_level Sub STRENGTH			254	114	128	N

The are two S meter records (RxA and RXb) and eight Tx meter records (reflected power not available on FTdx101D)

The S meter **code** values **SMTA** and **SMTB** are fixed in code and so must be entered as these so that the server will recognise them.
The Tx meter **code** fields are copied to the client at startup and so you can use any **code** of your choice.
Five of the Tx meters have been assigned to the available five option buttons as shown below



In the **meterhl** table, these are the items with a non-zero **btnno** field (ie: 61 to 65)

Meter table field list:

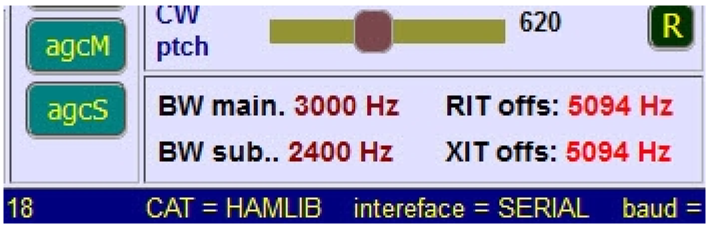
- **rig** The current radio - drop down selector (from radios table)
- **description** Descriptive text- no function
- **caption** The caption to the left of the slider. Note that this only applies to the central column of sliders with no adjacent on/off button with identifying caption.
- **btnno** The sliders's unique, fixed, numeric identifier as discussed earlier.
- **code** Specified once here. Used in message from client and recognised in server.
- **abx** If **A** or **B,** Client sends Vfo **A** or **B** to server. **abx** allows server to read appropriate meter.
  For Tx meters abx **= X** (not VFO-specific)
- **readmask** The character string used to read from the rig via rigctld.
- **setmask** The character string used to write to the rig via rigctld.
  Contains # which is substituted on button action by the data from **buttonshl.**
- **answermask** *This is used for commands not supported by rigctld where we use \send_cmd_rx.*
  It is the character pattern of the answer and has the same format as in the ASCII system.
  See Command masks
  We use **\send_cmd_rx** here because not all the metering levels are supported by **rigctld.**
- **mult** Multiplier (default = 1)
- **divide** Divisor (default = 1) Meter CAT value is scaled by (CAT * mult) / divide
  before being optionally modified by the **metercal** table data.
- **usecal** Y or N. If Y, then the CAT value is processed by the calibration table.
  (20 point calibration for each meter with linear interpolation between points)

Four of the records in the illustrated table use **\send_cmd_rx**.
These are explained in the next section : Unsupported commands

8.12 Hamlib -- Text display box - bandwidth display with WRXA and WRXB

A late modification was to replace the lowermost slider in the middle section by a text display box.
The box has four text data items, each with a caption in black and data text in a user defined colour.
The four text items appear in the **sliders** / **slidersciv** /**slidershl** tables with slidernos: **51** to **54**.

Each item is essentially a slider caption and a slider text data display without the slider in between them.
The caption is defined by **slider.caption** The **slider.color field** is used for the data text colour.
The **mult**, **divide**, **offset**, **units**, **lookup** and **decpoint** fields act on the data display exactly as for sliders.
min and **max** are unused. (They are used by sliders to match slider travel to min and max data range limits.)



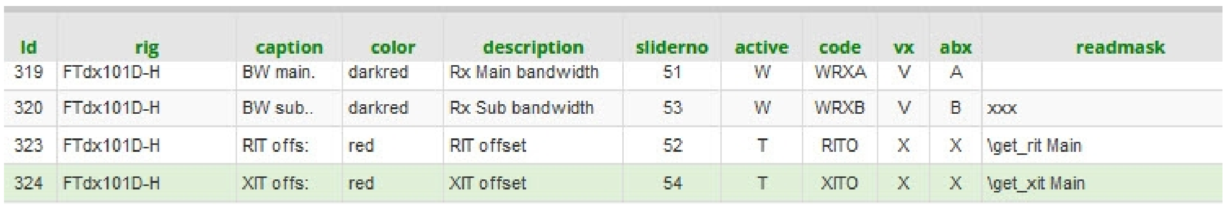
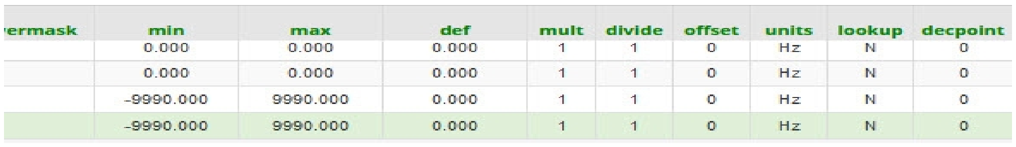
BW main. 3000 Hz RIT offs: 5094 Hz
BW sub.. 2400 Hz XIT offs: 5094 Hz
18 CAT = HAMLIB intereface = SERIAL baud =

In this example, the **RIT** and **XIT** items, 52 and 54 are synchronised to the data values in the rig.

The BW items, 51 and 53 here are using a special Hamlib feature. This is related to the fact that the necessary repetitive Hamlib **mode** readings also automatically return bandwidth.

This example is from my FTdx101-H Hamlib configuration (ie: connecting using the Hamlib **rigctld** daemon.)

The CAT data configuration is the same as for a slider (except **min** and **max** unused)
So please refer to the configuration information for table **slidershl**.

Id	rig	caption	color	description	sliderno	active	code	vx	abx	readmask	min	max	def	mult	divide	offset	units	lookup	decpoint
319	FTdx101D-H	BW main.	darkred	Rx Main bandwidth	51	W	WRXA	V	A		0.000	0.000	0.000	1	1	0	Hz	N	0
320	FTdx101D-H	BW sub..	darkred	Rx Sub bandwidth	53	W	WRXB	V	B	xxx	0.000	0.000	0.000	1	1	0	Hz	N	0
323	FTdx101D-H	RIT offs:	red	RIT offset	52	T	RITO	X	X	\get_rit Main	-9990.000	9990.000	0.000	1	1	0	Hz	N	0
324	FTdx101D-H	XIT offs:	red	XIT offset	54	T	XITO	X	X	\get_xit Main	-9990.000	9990.000	0.000	1	1	0	Hz	N	0

The above table is split into left and right parts. The unused **setmask** and **answermask** fields are not shown.
(Note that **answermask** would be used with **\send_cmd_rx** for a CAT command that is not supported by Hamlib)

The **BW main** and **BW sub** above are a special Hamlib feature, see below.
*The **RIT** and **XIT** items represent the usual configuration of these text data items for all other level data.*

RIT and XIT offset

The CAT data field here is **readmask** which with Hamlib **rigctld** is: **\get_rit Main.** (See: rigctl documentation)
The offset frequency data is returned as a numeric with range -9990 to +9990.
This needs no scaling and so **mult = 1** and **divide = 1**.
(For AF gain with range 0.000 to 1.000 we might set mult = 100 to scale the displayed value to 0 - 100)

Above, I have used **code=RITO** and **code=XITO**. I could have used any **code** of my choice.
code and **vx** are sent to the client. **code** and **abx** are used on the server. They form the client - server link.
(See: README - vx and abx)

active = T

For a text data item, we set **active = T** as in the above table.
This has the same effect as setting **active = S** (sync) for a slider.
This is essential so that the text item is repetitively updated with other controls with **active = S or T**.

8.13 Hamlib -- mode and bandwidth

rigctl uses commands **\get_mode** and **\set_mode** commands for both the **mode** and the **IF width** settings.
This adds a degree of complexity which I have tried to minimise in the configuration system.

The format for \set_mode is: **`\set_mode Main LSB 2700`**
`\get_mode Main` returns **LSB\n2700\n** (\n is newline, ASCII 10)

Reading mode

Mode reads for Main (or VFOA) and Sub (or VFOB) from the rig each have a dedicated repetitive process.
The repetition intervals for these reads are set in the **timing** table using fields:
modecur (currently selected VFO) and **modezz** (sleeping VFO)
Mode display is thereby continually updated, which means: highlighting the correct mode button, and updating the mode indicators to the left of the frequency displays.
If your chosen **IF width** slider has **code=IFWD**, then the bandwidth returned with **mode** will update the slider and its adjacent display (for the currently selected VFO).
The returned mode and bandwidth data also update stored: modeA, modedB, passbandA and passbandB.

Note the mode updates from the rig occur in the background and in Hamlib will update the IF width slider.
There is therefore no need to have the slider configured for sync. (So I use **active=Y** rather than active=S)
There is no need in fact to configure the **readmask** field for IF width. I set **readmask = xxx** which make the server return a XXX_COMMAND error code. This ensures no further action.

Setting mode is more tricky. We have to send both **mode** and **passband**.

On setting passband (ie: on moving the **IF width slider**), piWebCAT sends the existing **mode** from stored modeA or modeB and the new **passband** value derived from slider position.

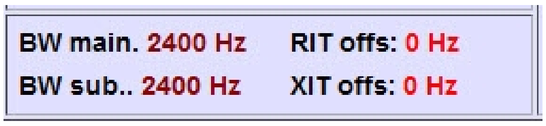
We use setmask = **\set_mode Main #.** This is used as follows:
The slider position derived bandwidth arises on the client. The current mode is held on the client.
The client software combines these into a string of the form: **LSB 2800**. This is the data sent to the server.
The server reads **\get_mode Main #** from the database and substitutes the data string for the # character.
This is then sent as the command to rigctld: **\get_mode Main LSB 2800**

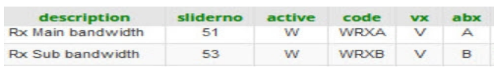
On setting mode (ie: clicking a **mode button**), piWebCAT sends the new **mode** and a ***passband value of -1***.
eg: **`\set_mode Sub CW -1`** . **`CW`** is from the CW button's **nset** value in the **buttonshl** table.
The **passband = -1** causes the radio to use the previously used bandwidth from the last time the mode was used.
(I could have sent the current stored passband .... but that would be for the old mode and so not a good idea.)
Sending **0** instead of **-1** selects the rig's default passband for the mode. if you prefer that, simply change -1 to 0)

Thus: both mode and IF width commands have to use **\get_mode** and **\set_mode**.
These are in **catcodeshl** for mode (button controlled) and in **slidershl** for IF width (slider controlled)

This combination of passband and mode into a single command is a Hamlib feature. I can see why it is done.
... but it made my life quite a bit mode complicated .... and yours a little more complicated!

Hamlib rigctld - Bandwidth display





description	sliderno	active	code	vx	abx
Rx Main bandwidth	51	W	WRXA	V	A
Rx Sub bandwidth	53	W	WRXB	V	B

In the text box and associated **slidershl** records above and on the previous page,
the **BW Main** and **BW Sub** items use a feature that is only available using Hamlib.

Mode data read events are set up in the **buttonshl** and **catcodeshl** tables as described above.
When the bandwidth is returned with **mode** data, piWebCAT checks for the existence of **sliderhl** text item records
with **active=W** and with **code=WRXA** (for Main/VFOA) or **code=WRXB** (for Sub/WRXB). (as above example)
The bandwidth data display is then updated.
(Main and Sub are Yaesu FTdx101D terminology. Most rigs use VFOA and VFOB)

8.14 Hamlib -- rigctld - Unsupported commands - \send_cmd_rx

Many radios have hundreds of less commonly used commands and configuration setting that are not supported by rigctl / rigcltd.

To deal with these, Hamlib provides the command: **\send_cmd_rx** to which we append CAT control sequences from the specific radio's CAT manual.

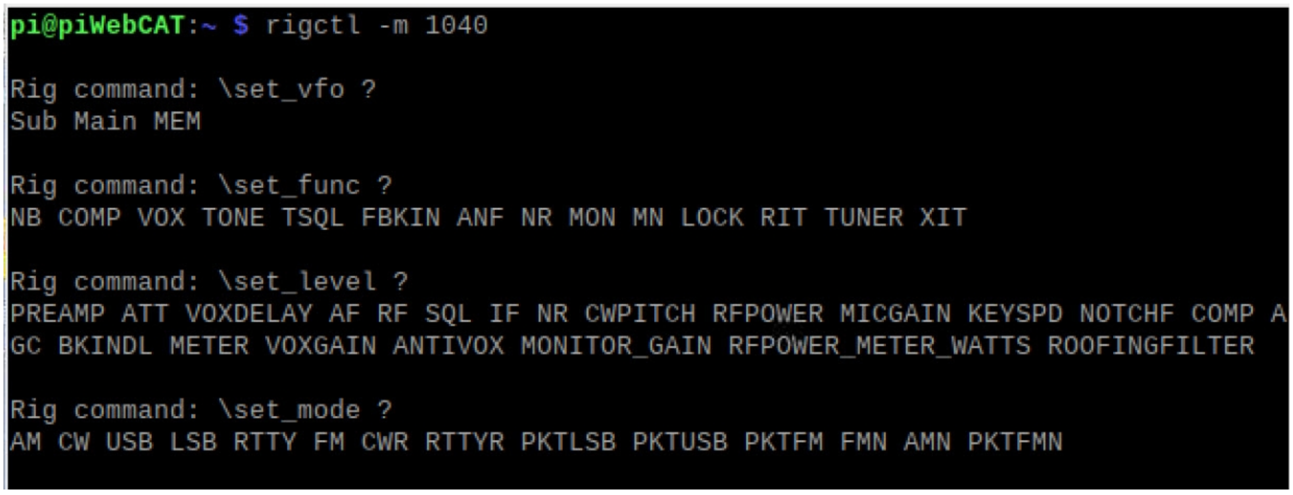
At the time of writing this, I have only used \send_cmd_rx as follows:

- *Fully implemented with an ASCII text based rig, the FTdx101D.*
- *Use for data write with a CIV command (Simplex duplex switching)*
  The rigctld documentation shows that binary data is supported using format \0xAA\0xBB.. etc

*However, piWebCAT would need special provision to and interpret data **received** back from an Icom CI-V rig. ie: bespoke processing code which I have not written.*

I illustrate in following sections the use of **\send_cmd_rx** with FTdx101D features that **rigctl** does not support:
These are: **roofing filter** switching and **contour and APF** switching and control

In: rigctl: supported rig controls I described how to determine the list of supported commands for a radio.
The results for the FTdx101D are repeated below:



```
pi@piWebCAT:~ $ rigctl -m 1040

Rig command: \set_vfo ?
Sub Main MEM

Rig command: \set_func ?
NB COMP VOX TONE TSQL FBKIN ANF NR MON MN LOCK RIT TUNER XIT

Rig command: \set_level ?
PREAMP ATT VOXDELAY AF RF SQL IF NR CWPITCH RFPOWER MICGAIN KEYSPD NOTCHF COMP A
GC BKINDL METER VOXGAIN ANTIVOX MONITOR_GAIN RFPOWER_METER_WATTS ROOFINGFILTER

Rig command: \set_mode ?
AM CW USB LSB RTTY FM CWR RTTYR PKTLSB PKTUSB PKTFM FMN AMN PKTFMN
```

Examination of the **func** and **level** lists reveal no codes for contour and APF (Audio Peak Filter) control.

The FTdx101D has a **CO....** command which supports on/off and frequency control for both contour and APF.

We can use **rigctld \send_cmd_rx** to send such actual rig commands when they are not supported by Hamlib.
This includes the extensive range of menu settings in modern transceivers.
For example: The FTdx101D has an **EX....** command which supports over 230 menu settings.
*piWebCAT's high degree of user configurability allows it to access these settings both in direct ASCII mode and also through Hamlib using **\send_cmd_rx.***

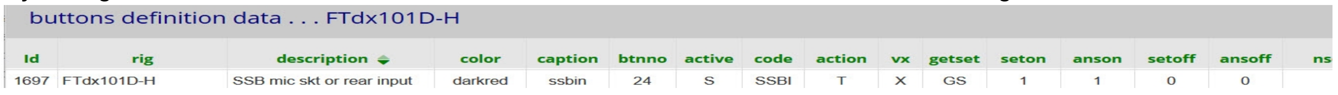
An example that I use is the menu option to switch SSB modulation audio source between front panel microphone socket and rear DATA connector. I need to switch to rear DATA connector source if I am remotely operating via the Mumble VOIP (Voice Over IP) system. The Yaesu command is: **EX010111#;** where # = **0** or **1**.
Using Hamlib to read this switch, I send:
\send_cmd_rx EX010111; 10 (The **10** informs piWebCAT how many returned characters to expect)
The button is labelled **ssbin** :

8.15 Hamlib -- unsupported commands - FTdx101D SSB audio source

The FTdx101D has an **EX....** command which supporst over 230 menu settings.
*piWebCAT's high degree of user configurability allows it to access these settings both in direct ASCII mode and also through Hamlib using **\send_cmd_rx.***

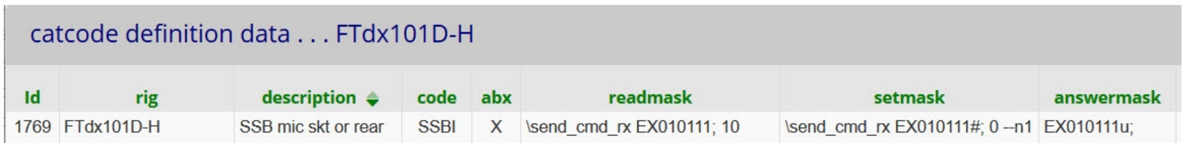
An example that I use is the menu option to switch SSB modulation audio source between front panel microphone socket and rear DATA connector. I need to switch to rear DATA connector source if I am remotely operating via the Mumble VOIP (Voice Over IP) system.

My configuration in **buttonshl** and **catcodeshl** for an SSB modulation source switching buttons are shown below:



buttons definition data . . . FTdx101D-H

Id	rig	description	color	caption	btnno	active	code	action	vx	getset	seton	anson	setoff	ansoff	ns
1697	FTdx101D-H	SSB mic skt or rear input	darkred	ssbin	24	S	SSBI	T	X	GS	1	1	0	0	



catcode definition data . . . FTdx101D-H

Id	rig	description	code	abx	readmask	setmask	answermask
1769	FTdx101D-H	SSB mic skt or rear	SSBI	X	\send_cmd_rx EX010111; 10	\send_cmd_rx EX010111#; 0 --n1	EX010111u;

In **buttonshl,** we have a single button. This has:

- **action = T** Toggle on/off.
  seton = 1 and setoff = 0 are set codes. anson = 1 and ansoff = 0 match received state data.
- **active = S** This set 'sync' mode so that the piWebCAT button is kept up to date with rig changes.
- **code = SSBI** An identifier of my choice - matches code in catcodeshl on the server.
- **vx = x** Not VFO dependant. Matches abx = X in catcodeshl on the server.

In **catcodedshl** we have the commands to communicate with the radio.

readmask and **sendmask** use ricgctl: **\send_cmd_rx.** This sends an appended *rig* CAT command.

Reading data from the rig

The FTdx110D CAT read command is : **EX010111;** (see FTdx101D CAT manual page 9)

As in the **catcodeshl** table above, the **readmask** field is **`\send_cmd_rx EX010111; 10`**
The appended **10** informs rigctld to expect to receive 10 bytes back from the rig (includes the semicolon)

rigctld will send **EX010111;** unmodified to the rig.
The rig will send back **EX010111n;** (10 bytes) where n= 1 for read DATA connector and n = 0 for mic. socket.
rigctld passes the **EX010111n;** unmodified back to piWebCAT.

The **answermask** field decodes the result. It is EX010111u; where **u** means units **-** ie: implying a single digit of data.
This format is exactly as is used to decode data when piWebCAT is in ASCII mode.

See Command masks on ASCII

Writing data to the rig

The **setmask** field is: **\send_cmd_rx EX010111# 0 --n1**

The # character is substituted by the data value from the client. This is from the **buttonshl** record's **nset** field.

For a read panel audio input, **nset = 1.** rigctld therefore sends **EX0101111;** to the rig.

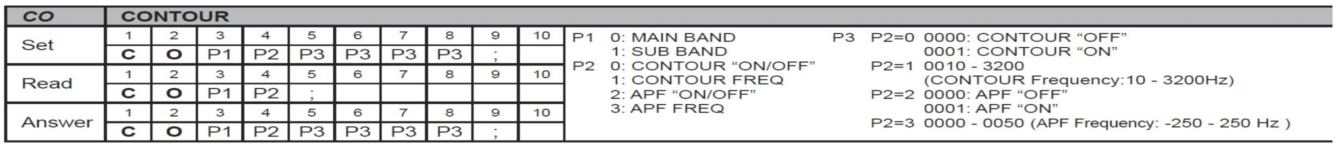
The appended **0** informs rigctld that there will be **0** bytes of response.
This is a rigctld requirement. (In some rigs, a set command will return data.)

The final **--n1** informs the piWebCAT PHP server code that the # character is substituted by **1** character.
This is a piWebCAT requirement.

8.16 Hamlib -- unsupported commands - FTdx101D contour and APF

In the FTdx101D, **rigctl** does not support Yaesu's contour and APF features. I therefore used **\send_cmd_rx**
The rig has separate contour and APF on/off switches but frequency adjustments share a single rotary control.
These switches and controls are duplicated for the two receivers.
On SSB, I can use contour. On CW, I can use contour or CW but not both.

From the FTdx101D CAT manual:
The **CO..** command provides both contour and APF on /off switching and contour and APF frequency adjustment.



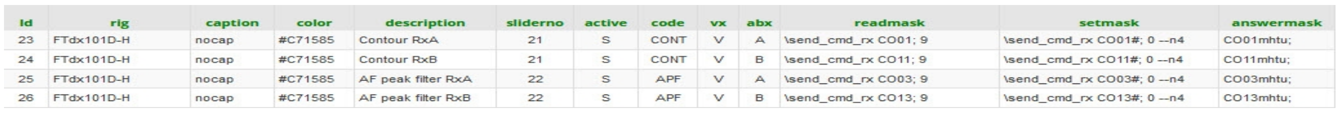
CO	CONTOUR											
Set	1	2	3	4	5	6	7	8	9	10	P1 0: MAIN BAND 1: SUB BAND P2 0: CONTOUR "ON/OFF" 1: CONTOUR FREQ 2: APF "ON/OFF" 3: APF FREQ	P3 P2=0 0000: CONTOUR "OFF" 0001: CONTOUR "ON" P2=1 0010 - 3200 (CONTOUR Frequency:10 - 3200Hz) P2=2 0000: APF "OFF" 0001: APF "ON" P2=3 0000 - 0050 (APF Frequency: -250 - 250 Hz)
	C	O	P1	P2	P3	P3	P3	P3	;			
Read	1	2	3	4	5	6	7	8	9	10		
	C	O	P1	P2	;							
Answer	1	2	3	4	5	6	7	8	9	10		
	C	O	P1	P2	P3	P3	P3	P3	;			

Contour frequency

For Main receiver, P1 = 0. For contour frequency, P2 = 1. P3 = four digit frequency, (**nnnn** below)
So all Main receiver contour frequency commands begin: **CO01.....**

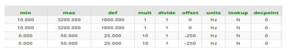
Read request is: **CO01;** Set request is: **CO01nnnn;** Response data string is **CO01nnnn;**

The configuration for contour and APF frequency in the **slidershl** table is shown below (split because of width):



Id	rig	caption	color	description	sliderno	active	code	vx	abx	readmask	setmask	answermask
23	FTdx101D-H	nocap	#C71585	Contour RxA	21	S	CONT	V	A	\send_cmd_rx CO01; 9	\send_cmd_rx CO01#; 0 --n4	CO01mhtu;
24	FTdx101D-H	nocap	#C71585	Contour RxB	21	S	CONT	V	B	\send_cmd_rx CO11; 9	\send_cmd_rx CO11#; 0 --n4	CO11mhtu;
25	FTdx101D-H	nocap	#C71585	AF peak filter RxA	22	S	APF	V	A	\send_cmd_rx CO03; 9	\send_cmd_rx CO03#; 0 --n4	CO03mhtu;
26	FTdx101D-H	nocap	#C71585	AF peak filter RxB	22	S	APF	V	B	\send_cmd_rx CO13; 9	\send_cmd_rx CO13#; 0 --n4	CO13mhtu;

There is no caption because the adjacent contour on/off switch has the 'Cont' caption.
vx = V so that the client sends current VFO, A or B.
abx = A or B for Main and Sub receivers.
See: vx and abx



min	max	def	mult	divide	offset	units	lookup	decpoint
10.000	3200.000	1600.000	1	1	0	Hz	N	0
10.000	3200.000	1600.000	1	1	0	Hz	N	0
0.000	50.000	25.000	10	1	-250	Hz	N	0
0.000	50.000	25.000	10	1	-250	Hz	N	0

Reading contour frequency

For Main receiver we have: **readmask** = `\send_cmd_rx CO01; 9`.
The **9** informs rigctld to expect a **9** character response.

The **answermask** field is **`CO01mhtu;`**
Typical received data might be: **CO010688;** where the data (frequency) is 0688

I use the ASCII configuration software. **`mhtu`** maps to decimal digits: thousands, hundreds, tens and units.
See Command masks on ASCII

Setting contour frequency

For Main receiver, we have setmask = **`\send_cmd_rx CO01#; 0 --n4`**
The **0** informs rigctld to expect no repsonse, ie **0** characters.
The **`--n4`** informs piWebCAT that the # is to be substituted with **4** data digits.

piWebCAT scales slider position (internally 0 - 400) to the range **10 - 3200**
(**min** and **max** fields are 10 and 3200)

`--n4` causes leading zeros to be added when necessary to achieve a **4** digit data string
eg: **345** from slider scaling becomes **0345** and so piWebCAT sends to rigctld: **`\send_cmd_rx CO010345; 0`**

APF frequency

This is handled similarly to contour frequency, but note the following differences.

The APF frequency range on the rig is **- 250 to + 250 Hz.**
The CAT range is **0 to 50.** We set **min = 0** and **max = 50** (**min** and **max** control scaling to and from the ***slider***)
(Note that some FTdx101D commands return negative data using **-** *and* **+** *chars. eg: IF shift -1200 to +1200 Hz. piWebCAT decodes these with the* ***s*** *(sign) character in the mask. This is not needed here.)*

Fields: **offset, mult, divide**, **decpoint** and **units** (and **lookup**) are only for the adjacent numeric display.

mult = 10 and **divide** =1 scales **0 - 50** to **0 - 500**

offset = -250 then scales **0 - 500** to **-250 to + 250**

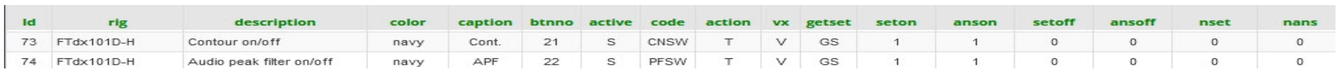
units = Hz to give a final display of eg: **-125 Hz**

Contour and APF on / off switching

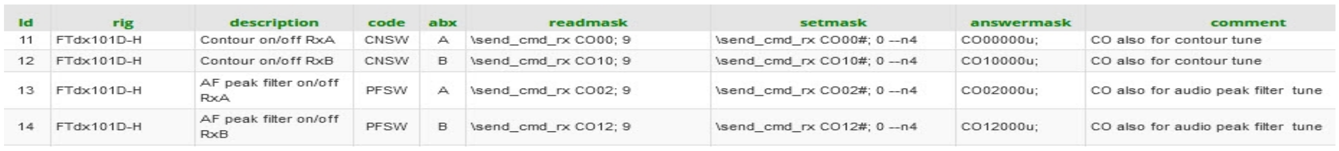
From the CAT manual table above:
Read Sub recvr contour on/off is **CO10;** **R**esponse is: CO10nnnn; nnnn = 0000 for OFF and 0001 for ON.
Read Main recvr APF on/off is **CO02;** **R**esponse is: CO02nnnn; nnnn = 0000 for OFF and 0001 for ON.

Setting Sub contour on/off status is **CO12nnnn**, where nnnn = 0000 for OFF and 0001 for ON.

The **buttonshl** and **catcodeshl** records are shown below:



Id	rig	description	color	caption	btnno	active	code	action	vx	getset	seton	anson	setoff	ansoff	nset	nans
73	FTdx101D-H	Contour on/off	navy	Cont.	21	S	CNSW	T	V	GS	1	1	0	0	0	0
74	FTdx101D-H	Audio peak filter on/off	navy	APF	22	S	PFSW	T	V	GS	1	1	0	0	0	0



Id	rig	description	code	abx	readmask	setmask	answermask	comment
11	FTdx101D-H	Contour on/off RxA	CNSW	A	\send_cmd_rx CO00; 9	\send_cmd_rx CO00#; 0 --n4	CO00000u;	CO also for contour tune
12	FTdx101D-H	Contour on/off RxB	CNSW	B	\send_cmd_rx CO10; 9	\send_cmd_rx CO10#; 0 --n4	CO10000u;	CO also for contour tune
13	FTdx101D-H	AF peak filter on/off RxA	PFSW	A	\send_cmd_rx CO02; 9	\send_cmd_rx CO02#; 0 --n4	CO02000u;	CO also for audio peak filter tune
14	FTdx101D-H	AF peak filter on/off RxB	PFSW	B	\send_cmd_rx CO12; 9	\send_cmd_rx CO12#; 0 --n4	CO12000u;	CO also for audio peak filter tune

Each has a **buttonshl** record with **vx = V** (send VFO A or B) and two **catcodeshl** records with **abx = A** and **B** for Main and Sub.

Reading Main recvr. contour on/off status:

I use: **readmask** = **`\send_cmd_rx CO00; 9`**. The **9** informs rigctld to expect a **9** character response.

The **answermask** field is **`CO01000u;`** Mask data characters are 000u - u = units
Data is returned as 0000 or 0001.

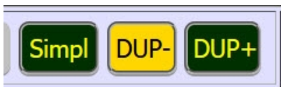
See Command masks on ASCII

Setting Main recvr. contour on/off status:

I use: **setmask** = **`\send_cmd_rx CO00#; 0 --n4`**
The **0** informs rigctld to expect no response, ie: a **0** character response.
The **`--n4`** informs piWebCAT that the **#** is to be substituted with **4** data digits. (ie: 0000 = OFF or 0001 = ON)

8.17 Hamlib -- unsupported commands IC7000 duplex /simplex switching

Using **\set_level ?** and **\set_func ?** at the **rigctl** command line reveals that **rigctl** does not support IC7000 duplex / simplex switching commands.
I therefore configured the buttons below using rigctld **\send\cmd_rx** which appends actual rig CAT commands.



The rig doesn't have the means to assign repeater operation to specific channels and so Duplex - or + has to be selected. piWebCAT's buttons here are much faster and easier to access than the corresponding control sequence on the rig.

From the **rigctl** documentation, binary data (eg: for Icom CI-V) is entered as: **`\send_cmd_rx \0xAA\0xBB`**

From the IC7000 manual CAT section:

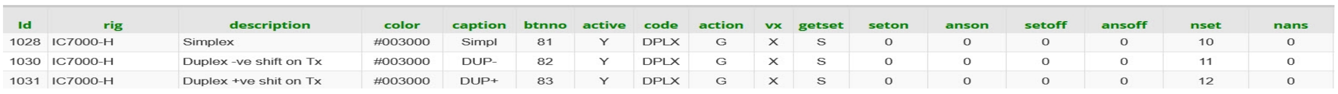
With **command = 0x0E:**
- Subcommand **0x10** selects simplex operation.
- Subcommand **0x11** selects **`-DUP`** operation.
- Subcommand **0x12** selects **`+DUP`** operations.

These are Icom CI-V **set** commands with no other associated data.

The full CI-V command string for +DUP is: **`0xFE 0xFE 0x70 0xE0 0x0E 0x12 0xFD`**
(0xFE 0xFE and 0xFD are obligatory fixed bytes. 0x70 is the rig address. 0xE0 is a sender address 0x0E is the command and 0x12 is our subcommand for +DUP.)

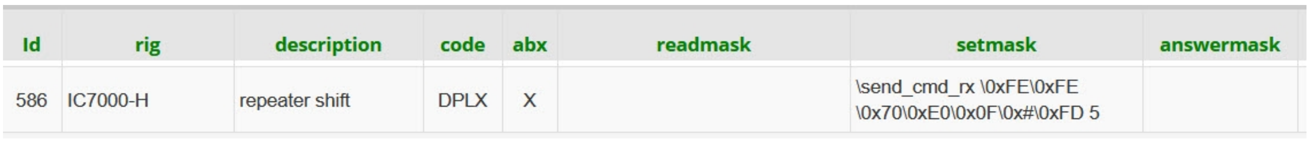
Reading simplex/duplex status ?
A scan of available commands finds no commands to ***read*** Duplex/Simplex status.
Synchronisation of button to rig at startup is not therefore possible here.
Furthermore, Interpreting data returned from \send_cmd_rx would require bespoke software because to the complex Icon CI-V formats, I haven't produced this!

The configuration tables are shown below:
buttonshl



Id	rig	description	color	caption	btnno	active	code	action	vx	getset	seton	anson	setoff	ansoff	nset	nans
1028	IC7000-H	Simplex	#003000	Simpl	81	Y	DPLX	G	X	S	0	0	0	0	10	0
1030	IC7000-H	Duplex -ve shift on Tx	#003000	DUP-	82	Y	DPLX	G	X	S	0	0	0	0	11	0
1031	IC7000-H	Duplex +ve shit on Tx	#003000	DUP+	83	Y	DPLX	G	X	S	0	0	0	0	12	0

catcodeshl



Id	rig	description	code	abx	readmask	setmask	answermask
586	IC7000-H	repeater shift	DPLX	X		\send_cmd_rx \0xFE\0xFE \0x70\0xE0\0x0F\0x#\0xFD 5	

I have chosen **code=DPLX**. **code** is the link between server data (**catcodeshl**) and client data (**buttonshl**)
(I could have used any **code** of my choice here as there are no restrictions from internal uses of the **code**.)

The three buttons are grouped (shared **code = DPLX** and **action = G**).
Grouped buttons use the **nset** and **nans** field for data. We use only **nset** here because we only have **set** functionality.
From the IC7000 CAT information, the three values are **0x10**, **0x11** and **0x12**.
The fields are actually held as text. I have used **10**, **11** and **12** as the **nset** values**.**

In **catcodedshl**, the **setmask** is **`\send_cmd_rx \0xFE\0xFE\0x70\0xE0\0x#\0xFD 5`**

The # character, as in all **rigctld** commands is substituted by the data from the client records (**buttonshl)**

Thus **`0x#`** becomes **`0x10`**, **`0x11`** and **`0x12`** for the three buttons.

8.18 Generating a Hamlib error log for a problem / bug report

The Hamlib API is continually being refined by the development team.
WIth 250 rigs supported, I have observed a new code version appearing on the majority of days in the eight months that I have been developing piWebCAT with Hamlib.

The developers do not have 250 radios on which to test their code and so they are dependent on user feedback. When a new rig appears on the market, its Hamlib support has to be developed from its CAT manual together with the experience of similar rigs.
I have only four of the 250 supported rigs here and I am in a similar position with piWebCAT.

Generating the error log

Using piWebCAT's configuration editor, edit the **rigs** table.
Temporarily change the **hamlib** field from your rig's hamlib code to **zero**.
This will prevent piWebCAT, on start up from starting Hamlib **rigctld**.

On the RPi, open a terminal window.
Type: **`sudo killall rigctld.`**
(rigctld may still be running if you last exited piWebCAT by just exiting the browser)
Type: **`script logfilename.txt`** ... everything appearing in terminal will now be recorded.
Type the following (I give the example for my Ftdx101D)

```
rigctld -m 1040 -s 38400 -m /dev/ttyUSB0 --vfo -Z -vvvvv
```

- **1040** is the FTdx101D's hamlib number - substitute yours
- **38400** is my baudrate - substitute yours.
- **/dev/ttyUSB0** if for USB connection. For serial port, substitute **/dev/ttyAMA0**.
- **--vfo** sets VFO mode. Omit if not used (eg: with Icom)
- **-Z** dates the generated log items.
- **-vvvvv** With five v's this generates maximum log information.

rigctld will start on the terminal. You will see several lines of log generated.
If the green $ prompt returns, you have entered something wrong.
Now you have two options:

1. **Open a second terminal window.**
   Type **`telnet 127.0.0.1 4532`**
   You can now type in individual **rigctld** commands (eg: **`\get_freq Main`**) and:
   - observe their response on the following line **(set** commands will report ok as RPRT 0)
   - observe verbose log text generation in the first window.
2. **Start piWebCAT.** It will run using your **rigctld** in the terminal window.
   Startup will generate a lot of log text and then log generation will continue as piWebCAT continues its stream of background tasks.
   Don't let this run for too many seconds .. otherwise you will generate a huge log which is large to email and which someone has wade through.
   If your problem relates to a control operation or tuning action on piWebCAT, then get these test actions done quickly and then exit piWebCAT (to keep down the log size)
   Finally, type Ctrl-C at the terminal window to exit.

Don't forget to restore your hamlib rig number in piWebCAT's rig table !!
If you need both the telnet and piWebCAT logs, please generate them separately!

The log file, **logfilename.txt** will be on the RPI in folder **/home/pi**.
Use FileZilla (other other FTP client) to connect to this folder.
The preconfigured FTP access to **/home/pi** is:
host = **192.168.1.117** (or your IP address) port = **21** user = **piuser** password = **feline**

Download the logfile(s) to your PC.

If you have used the telnet option 1. above, you may wish to also have a copy of your telnet dialog with rigctld. This can be done easily if you used Real VNC access on your PC. Expand the terminal window vertically to show all your text. Select it all with the mouse. Then use **menu - Edit - Copy**. Then paste it into an open instance of Notepad and save.
Yes - using RealVCN, you can copy and paste text between the RPi and the PC.

9.1 Meter background images

piWebCAT has:

- An S meter serving the receiver on the currently selector VFO
- Five buttons to select one of five Tx metering options.
  (The radio may have more than five Tx metering options, but you have to choose
  which to assign to the five buttons - see meter table or meterciv table)

piWebCAT is supplied with six meter background images:

• S meter	cat/images/smeter.bmp	
• Power output meter	cat/images/TXmeterA.bmp	left button
• ALC meter	cat/images/TXmeterB.bmp	
• Speech comp. meter	cat/images/TXmeterC.bmp	middle button
• PA IDD meter	cat/images/TXmeterD.bmp	
• SWR meter	cat/images/TXmeterE.bmp	right button

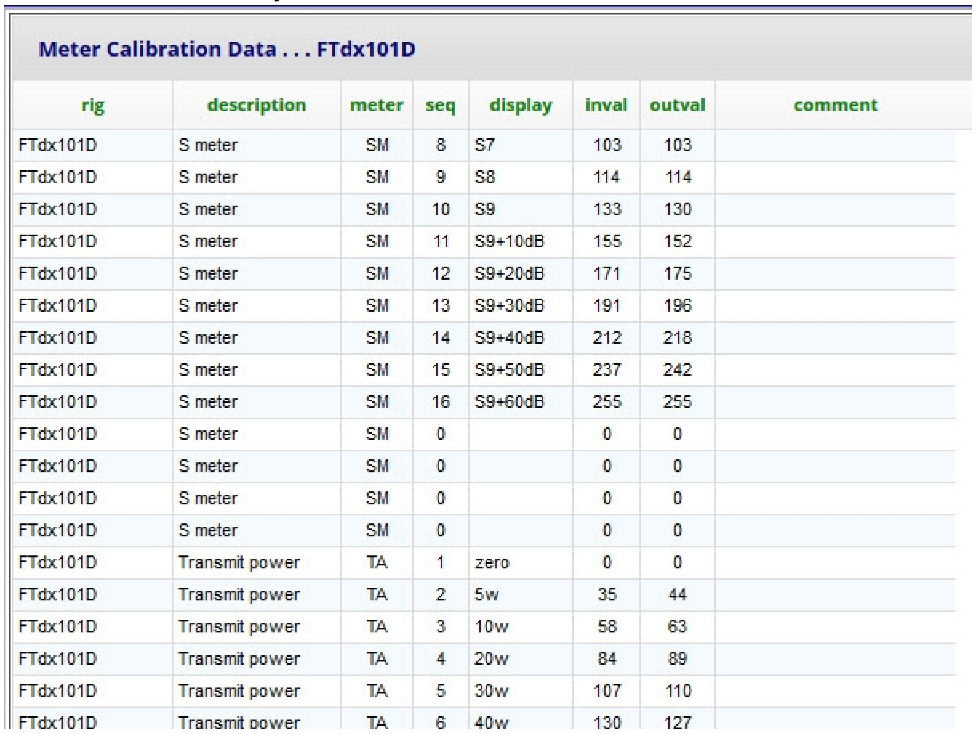
Each image is a 250 x 110 pixel bitmap.

If you want to display some other Tx value, then you can replace one or more of the bitmaps.
(You must of course use filenames TXmeterA.bmp etc, otherwise piWebCAT will not find them.)

When making a new bitmap images, I strongly suggest that you keep the arc,
and that you position the markers and number in a similar way to my originals.
If you do this they will look ok with the red meter needle (which you cannot change!)

9.2 S meter and Tx meter calibration

Below is a section of my **metercal** table for the FTdx101D



Meter Calibration Data . . . FTdx101D

rig	description	meter	seq	display	inval	outval	comment
FTdx101D	S meter	SM	8	S7	103	103	
FTdx101D	S meter	SM	9	S8	114	114	
FTdx101D	S meter	SM	10	S9	133	130	
FTdx101D	S meter	SM	11	S9+10dB	155	152	
FTdx101D	S meter	SM	12	S9+20dB	171	175	
FTdx101D	S meter	SM	13	S9+30dB	191	196	
FTdx101D	S meter	SM	14	S9+40dB	212	218	
FTdx101D	S meter	SM	15	S9+50dB	237	242	
FTdx101D	S meter	SM	16	S9+60dB	255	255	
FTdx101D	S meter	SM	0		0	0	
FTdx101D	S meter	SM	0		0	0	
FTdx101D	S meter	SM	0		0	0	
FTdx101D	S meter	SM	0		0	0	
FTdx101D	Transmit power	TA	1	zero	0	0	
FTdx101D	Transmit power	TA	2	5w	35	44	
FTdx101D	Transmit power	TA	3	10w	58	63	
FTdx101D	Transmit power	TA	4	20w	84	89	
FTdx101D	Transmit power	TA	5	30w	107	110	
FTdx101D	Transmit power	TA	6	40w	130	127	

The table contains 120 records per radio = 20 calibration points for each of six meters.
I strongly suggest that you do not need to modify the layout.
Simply enter the calibration data for you radio's selected meters.

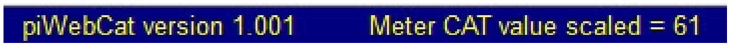
If the meter falls back to zero at full scale, then add a final extra record:
eg: seq = 17 inval = 255 outval = 255

The S meter has **meter** = SM. The Tx meters have **meter** = TA, TB, TC, TD and TE.
(Note that meter function and button captions are defined in **meter** or **meterciv** table.)

In the **meter** and **meterciv** tables, each meter has **mult** and **divide** for optional linear scaling
This scaling can be used for pre-calibration (eg 0-255 to 0-100)
This scaling happens before the calibration table is applied.
The meter CAT data on the page footer is AFTER the scaling.

Each meter can have up to 20 calibration points.
The seq field defines the order (needed for interpolation), Unused points have seq = 0.

Calibration procedure: uses the CAT value on the footer bar



piWebCat version 1.001 Meter CAT value scaled = 61

The meter's **usecal** field in the meter or **meterciv** table must be set to N (= no!).
Calibration needs steady meter levels
eg: S meter - use RF gain control. Tx power - CW to dummy load.

Example - S meter:
- first entry inval = 0 outval = 0
- then set S1 level on the radio and record CAT data for inval.
- then set S1 level on piWebCAT and record CAT data for outval
- repeat for S2 ... S9 and +10dB ... + 50dB etc.
- Enter the collected values into the metercal table as above.

Finally set **usecal** in **meter** or **meterciv** table to **Y** to test and use the new calibration.

10.1 Receive and transmit audio - Mumble

Introduction

With a web based system, an audio link for receive and transmit audio allows the use of an operating position remote from the transceiver. The technology is referred to as VOIP (Voice Over Internet Protocol).
VOIP was a late addition to piWebCAT in mid January 2021.
I searched the internet for suitable bidirectional VOIP packages and chose to use **Mumble.**
Mumble is a free, open source. low latency, high quality voice chat application.
The Mumble website is: https://www.mumble.info

Mumble uses a VOIP server which can communicate with multiple clients.
Client software is available for Windows, Linux, Android and Apple IOS.
Mumble is not part of piWebCAT. It operates alongside piWebCAT but is separate from it.

Other suitable VOIP applications could probably be used with piWebCAT.
I needed a VOIP application to present piWebCAT with remote operating capability and Mumble does the job.

In learning and evaluating Mumble, I fairly quickly achieved Rx and Tx audio communication between my FTdx101D and my Levono Yoga 710 laptop, a 10 inch Android tablet and my S6 ageing S6 mobile phone.

Security

Mumble uses secure encryption for audio transmision.
Automatic generation of a security certificate is provided in the mumble client installation process.
A password is set in the server installation process and a matching password can be used in the client.
However, I use the option of operating with a secure certificate for encryption but no login / password process.
This allows auto start of Mumble processes on the RPi without any keyboard /mouse access.

In piWebCAT, I have already stated my opinion that security is not an issue when communicating with a transceiver.
However, if you disagree with this view then change to your own passwords.
(You may feel the need to do this if operating across the internet.)

Communication with the transceiver

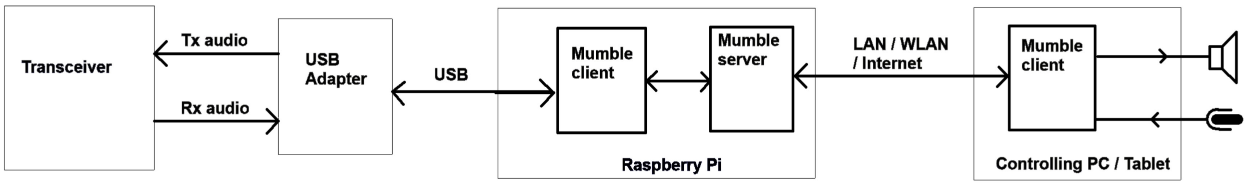
The RPi has an audio-out socket but no audio input.
I used an USB audio adapter purchased from PiHut for £4.50.
I made up a cable with two 3.5mm audio leads to connect to the data-in and data-out pins of the 6-pin RTTY/DATA socket on the rear of my FTdx101D.
A rig menu option selects SSB input from the rear connector instead of mic. socket.
Audio output is 300mV peak.
It is not controlled by the AF gain control - so it is not controlled by piWebCAT's AF gain slider.
I use the audio control on the controlling tablet or laptop.
The FTdx101D also has Rx audio available via USB ... which I can select in the RPi mumble configuration.
However, I can't locate speech input via USB, so it is simpler to use analog audio with the adapter for both Rx and Tx.

*Note that the Mumble audio pathway delays audio in both direction. This can be up to a second on receive. It makes tuning on receive feel slightly odd - until you get used it. In particular, SSB pitch change is slightly delayed
as you tune.*

10.2 Mumble system - structure and configuration

mumble-server is installed on the RPi.
mumble (client) is installed in the RPi and on the controlling device (PC, tablet etc)

The diagram below shows the system structure and data pathways



Mumble configuration.

Mumble is preinstalled in the RPi SD card image. The full installation procedure is detailed in section 14.6.
Part of the client installation uses post-installation settings which are also discussed here.

mumble-server has simple configuration options: **Auto start**, **high priority**, **password**s and your **welcome message**.

mumble-client initial configuration needs to specify the mumble-server **IP address** and **port.**
I use the IP address of the RPi (where the mumble-server resides) and the default mumble port = 64738.

mumble-client configuration options include the following:

- **Reconnect** to the server automatically. Reconnect to the last used server on startup.
- **Direct connection**. (No use of proxy - server and client are in the RPi)
- **Audio input** (Rx audio from transceiver)
  - Use ALSA (Advanced Linux Sound Architecture)
  - Use device: [hw:CARD=Device.DEV=0] PnP Sound Device, USB Audio Direct hardware without any conversions.
    (There exists a list of options. I chose here the USB audio adapter with *no signal processing*)
  - Compression quality 72 kb/s
  - Noise suppression off
  - Maximum gain
- Audio output (Tx audio to transceiver)
  - Use ALSA
  - Use device: [hw:CARD=Device.DEV=0] PnP Sound Device, USB Audio Direct hardware without any conversions.
    (Same as audio input.)
  - Volume100%. Output delay 50ms

The *'hardware without any conversions'* is important.
(Otherwise on a quiet channel, the background noise is suppressed to an unhelpful warbling noise.)

Automatic startup

The completely automatic startup of RPi based client and server is very important.
In normal operation, it allows the RPi to be treated as an autonomous **'black box'**
with no need for mouse and keyboard intervention.

mumble-server is configured for automatic starup and runs as a background service.

mumble client on the RPi is a visible (GUI) application running on the RPi LXDE desktop (Lightweight X11 Destop Environment).
It can be started from a desktop icon.
In order for it to start, automatically, it has to start after the LXDE is up and running.
The commonly used reboot start processes do not work here. I had to create a startup file: /etc/xdg/autostart/mumblestart.desktop.
Once running, the auto-connect options (as listed above) complete the autostart process.

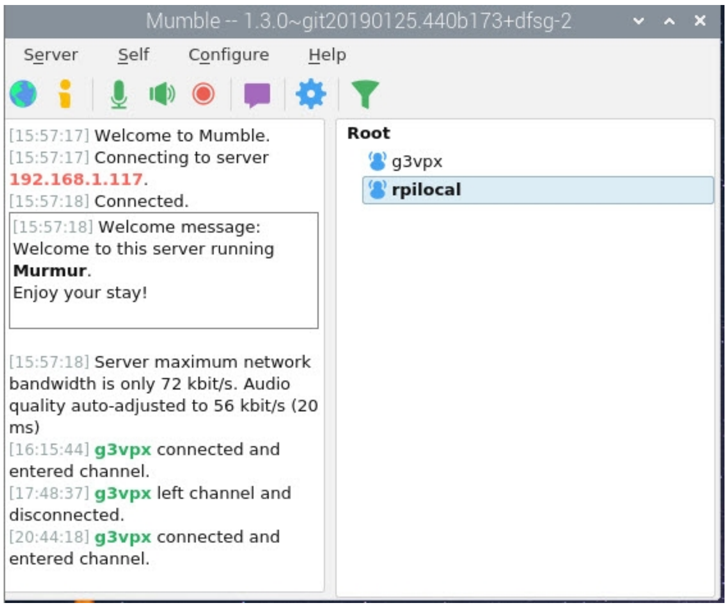
mumble-client on the controlling PC / tablet is started simply by clicking a desktop icon.

10.3 Mumble client on the RPi

The image below show the main window of Mumble running on the RPi.
The RPi IP address on my local network is 192.168.1.117.
The right hand panel shows:

- **rpilocal:** the username that I assigned to the local client on this RPi.
- **g3vpx:** the username of the mumble client on my controlling connected laptop.

The supplied micro SD card has IP address 192.168.1.117 and mumble client username = rpilocal.
The full installation procedure is detailed in section 14.6.



```
[15:57:17] Welcome to Mumble.
[15:57:17] Connecting to server 192.168.1.117.
[15:57:18] Connected.
[15:57:18] Welcome message:
Welcome to this server running Murmur.
Enjoy your stay!

[15:57:18] Server maximum network bandwidth is only 72 kbit/s. Audio quality auto-adjusted to 56 kbit/s (20 ms)
[16:15:44] g3vpx connected and entered channel.
[17:48:37] g3vpx left channel and disconnected.
[20:44:18] g3vpx connected and entered channel.
```

Root
- g3vpx
- **rpilocal**

Important

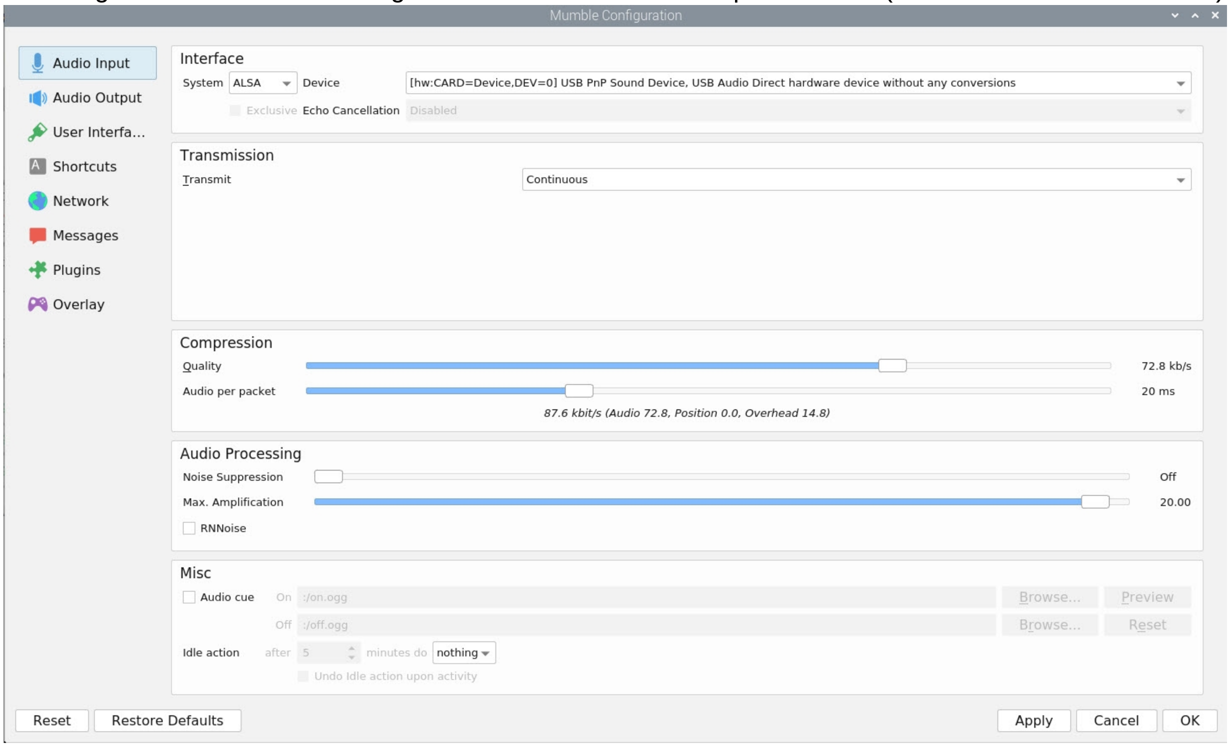
A feature of mumble:

Clicking Apply and then Ok does NOT save your changes.

You have to quit mumble and restart to save changes.

I wasted a few hours puzzling this !!

The image below shows the configuration window with Audio input selected (= Rx audio from transceiver)



Interface
System: ALSA Device: [hw:CARD=Device,DEV=0] USB PnP Sound Device, USB Audio Direct hardware device without any conversions
Exclusive Echo Cancellation: Disabled

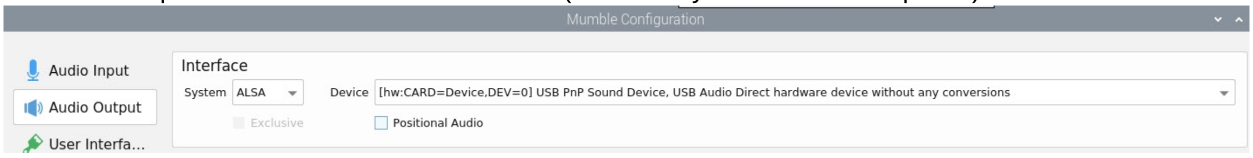
Transmission
Transmit: Continuous

Compression
Quality: 72.8 kb/s
Audio per packet: 20 ms
87.6 kbit/s (Audio 72.8, Position 0.0, Overhead 14.8)

Audio Processing
Noise Suppression: Off
Max. Amplification: 20.00
RNNoise

Misc
Audio cue On :/on.ogg
Off :/off.ogg
Idle action after 5 minutes do nothing
Undo Idle action upon activity

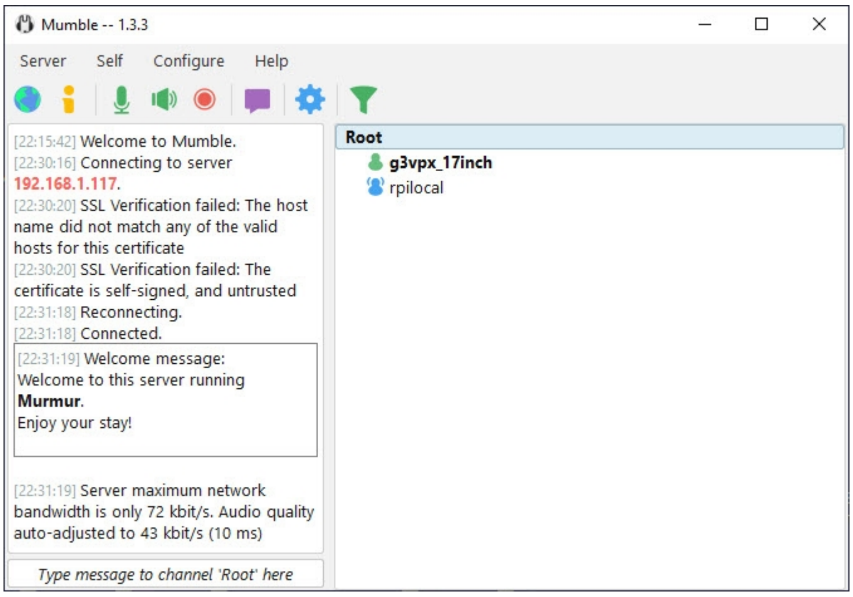
The audio output has the same device selection (from a very different list of options)



Interface
System: ALSA Device: [hw:CARD=Device,DEV=0] USB PnP Sound Device, USB Audio Direct hardware device without any conversions
Exclusive Positional Audio

10.4 Mumble client on a Windows PC

The image below shows the main window of mumble running on the RPi.



[22:15:42] Welcome to Mumble.
[22:30:16] Connecting to server 192.168.1.117.
[22:30:20] SSL Verification failed: The host name did not match any of the valid hosts for this certificate
[22:30:20] SSL Verification failed: The certificate is self-signed, and untrusted
[22:31:18] Reconnecting.
[22:31:18] Connected.
[22:31:19] Welcome message:
Welcome to this server running **Murmur**.
Enjoy your stay!

[22:31:19] Server maximum network bandwidth is only 72 kbit/s. Audio quality auto-adjusted to 43 kbit/s (10 ms)

Important

A feature of mumble:

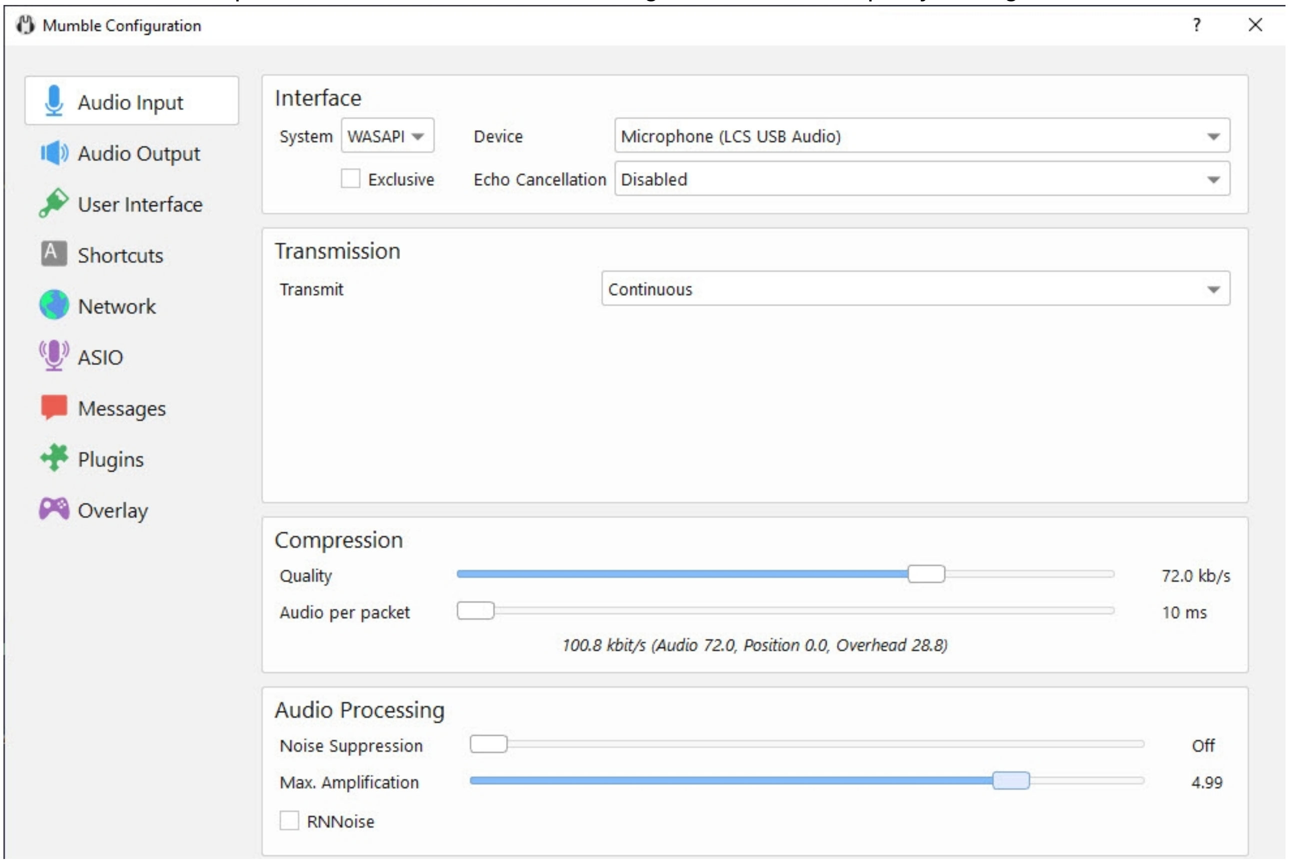
Clicking Apply and then Ok does NOT save your changes.

You have to quit mumble and restart to save changes.

I wasted a few hours puzzling this !!

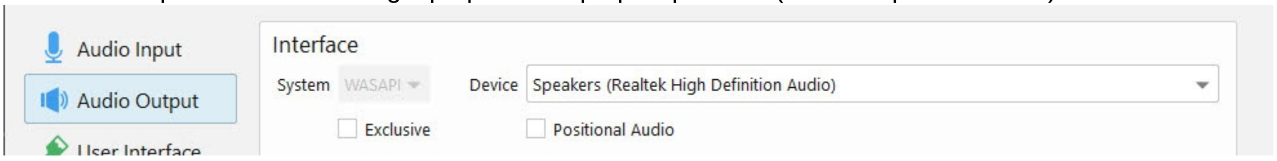
g3vpx_17inch is the user name assigned when setting up mumble client on this 17inch Lenovo laptop.
rpilocal is the user name of the mumble client on the RPi.

The image below shows the configuration window with Audio input selected (= Tx audio from a USB microphone) Note USB mic. as input device. Note **continuous** setting. Recommended quality setting is 72 kb/s.



System: WASAPI
Device: Microphone (LCS USB Audio)
Exclusive: ☐
Echo Cancellation: Disabled
Transmit: Continuous
Quality: 72.0 kb/s
Audio per packet: 10 ms
100.8 kbit/s (Audio 72.0, Position 0.0, Overhead 28.8)
Noise Suppression: Off
Max. Amplification: 4.99
RNNoise: ☐

The audio output on the controlling laptop is the laptop's speakers (and headphone socket)



System: WASAPI
Device: Speakers (Realtek High Definition Audio)
Exclusive: ☐
Positional Audio: ☐

10.5 Using mumble - some notes

All my development and testing of Mumble was done using my FTdx101D.

Audio source - receiver audio output

My FTdx101D has three possible sources of receiver audio to feed into the USB audio adapter.

- **Rear panel: The 6-pin data connector**. This appears to have output only from the Main receiver. It has a fixed level. *The front panel and CAT audio gain controls have no effect.*
- **Read panel: A 3.5mm stereo jack** socket with Main and Sub receiver outputs separate on the two pins . It has a fixed level. *The front panel and CAT audio controls have no effect*.
- **The front panel headphone socket**. This combines the output from both receivers. *The front panel audio gain controls and CAT audio gain controls are effective.*

Therefore, if you want to use Mumble remotely, with access to both receivers and have separate control of AF gain, then you have to use the headphone socket to feed the USB adapter.
In some radios, an attenuator might be needed.
In addition, piWebCAT's audio gain swapping feature (on VFO swap) can only work using CAT audio gain control and so would have to use the headphone socket.

Audio source - receiver microphone input

My FTdx101D has a menu option to select SSB microphone input between front panel microphone socket and rear DATA connector. Some CAT software packages do not offer this switching as a CAT option. However, this switching is available as a CAT command (EX0101110; EX0101111;) and so piWebCAT can be configured to control it, either with a toggling button or a pair of buttons (your choice).

Laptops / tablets - received audio quality

- Connection of RPi mumble client to transceiver.
  As stated in sections 10.2 and 14.6 it is important to install the USB audio adapter '*without any conversions'.* Otherwise on a quiet channel, the background noise is suppressed to an unhelpful warbling noise. In addition, I disable any noise reduction by Mumble.
- Audio quality is not helped by speaker quality on some devices.
- Some of the available configuration facilities on the mumble clients appeared to make no difference to audio quality and so my configuration has them unused or disabled.

Laptops / tablets - transmitted audio

- Microphones: The built in microphones are not as good as using an external microphone.
  I purchased an excellent USB desk microphone from Amazon (EIVOTOR £23).
  This produced very good signal reports.
  The microphone has a built in volume control which was a deliberate choice.
  The FTdx101D has mic. gain and processor adjustments (which are repeated in piWebCAT).
  The mic gain control is not effective when feeding audio into the rigs rear DATA/RTTY connector and so a gain control on the microphone is useful to optimise compression and ALC levels.
  (The configuration options in mumble client provide for preset level adjustment but are inconvenient for operational adjustment)
- I do not use Mumble's noise reduction and compression facilities.

Transmitted audio delay

There is a delay of perhaps 600ms between spoken audio and outgoing transmission.
I find that this makes real time self monitoring almost impossible. The delayed feedback has a bizarre effect on my speech!! I therefore used the rig's recording facilities to record test audio via Mumble.
I could then transmit this on dummy load and monitor on a separate receiver.

Received audio delay

There is a similar audio delay on received data.
Its effect on tuning the receiver is not really noticeable.
It is probably largely offset by the simultaneous small delay between the piWebCAT tuner and the response of the receiver.

11.1 Micro SD card features. piWebCAT website folder contents.

MicroSD card

Fully configured card image with:

- **RPi OC** (previously Raspian linux) - IP address set to a default of 192.168.1.112
- **Apache2 webserver**
- **MariaDB** (MySQL) database system - with the piWebCAT database:
  radios already configured and populated.
- **PHP7** server scripting language installed with database connectivity features added.
- **pureftpd** FTP server for remote file transfer to and from web root (by FileZilla and Expression 4)
- **phpmyadmin** installed to allow management of MySQL database from a PC
  (access as 192.168.1.112/phpmyadmin)
- **Real VNC server** activated to allow RPi keyboard/mouse/screen control using Real VNC viewer on a PC.
- Changes to **/boot/cmdline.txt** and **/boot/config.txt**
  - swap serial USARTs - the fast one to GPIO for piWebCAT.
    - the slow one for bluetooth (ok for mouse/kb)
- **Mumble** Voice Over IP server an client. (This autostarts on the RPi desktop)

Note the default RPi serial port behaviour is not good for pure binary data transfers needed by Icom CIV.
Some control characters and line feed are changed.
This is corrected in piWebCAT's serial port initialisation which switches the port to **RAW mode**:

In PhpSerial.php (GPL 2 licence, multiple authors) G3VPX added: **`function confRawMode()`**

piWebcat files and folders

The Raspberry Pi runs Linux OS, Apache2 web server, PHP7 server programming and MariaDB database.

The website root is located in the usual folder: **/var/www/html**
This contains:

- catcontrol.php - the main piWebCAT webpage.
- catedit.php - the database editor webpage
- metercal.php - the calibration page for the S meter and the Tx meters for each radio.
- index.php - the start page (initial access to http://server-address will load a file thus named)
- piwebcat.dwt - the Microsoft Expression template for the common parts of the three main pages
  ie header buttons with radio selector and the footer bar.
  (piwebcat.dwt is not needed for website operation)
- **cat/images folder** - six meter background bitmaps
- cat/jqueryplugins - jquery-ui-touch-punch.js
  - Translates touch screen actions into mouse events.(imported)
  - jquery-mousewheel.js - Support for the mousewheel. (imported)
- **cat/js folder** - piWebCat javascript files developed by G3VPX
- **cat/phpfiles folder** - piWebCAT php files developed by G3VPX
- **cat/popups folder** - popup windows - only message boxes at present
  - HTML code loaded by PHP HEREDOC system.
- **cat/css folder** - HTMP stylesheet files (slider.css and lines.css are imported)
- **phpgrid folder** - phpGrid (from HongKong). 3500 files - trimmed. (includes jQuery and jqGrid)
  G3VPX has an OEM distribution license.
- phhmyadmin - for PC database configuration access.
- **help folder** - complete help system - this manual

FTP access

For site configuration in FileZilla etc and web developer software (Microsoft Expression 4 or other)

Host = **192.168.1.117** port = **21** user = **upload** password = **feline**

11.2 Database configuration, users, password etc

Security is not needed when controlling your radio
.... therefore I have the same username and password for:
- internal localhost access (ie: piWebCAT) and
- external (port 3306) access by PC based **MySQL Front** software.

eg: MySQL Front: host = **192.168.1.117** port = **3306** user = **piwebcat** pw = **feline**

Internal database access for piWebCAT is defined in the following file:

file: cat/phpfiles/wcmysql.php:

```
<?php
   //  MYSQL access
   $DbServer = 'localhost';
   $DbUser = 'piwebcat';
   $DbPw = 'feline';
   $Database = 'radios';
?>
```

The built in database editor uses phpGrid.
file: phpgrid/conf.php is modified so that phpGrid uses the above definitions in wcmysql.php

ie: added as below (This is the ONLY change to the phpGrid code
.... apart from possible future removal of some unused code for bulk reduction)

```
require_once "/var/www/html/cat/phpfiles/wcmysql.php";

define('PHPGRID_DB_HOSTNAME',$DbServer); // database host name
define('PHPGRID_DB_USERNAME',$DbUser); // database user name
define('PHPGRID_DB_PASSWORD',$DbPw); // database password
define('PHPGRID_DB_NAME',$Database); // database name
define('PHPGRID_DB_TYPE', 'mysql'); // database type
define('PHPGRID_DB_CHARSET','utf8'); //
```

11.3 piWebCAT - main control page startup

This section assumes some prior reading of the database configuration sections.

Part of the data held in database tables is needed on the client.
eg for buttons: whether active, caption, colour, mode of action (toggled, grouped etc)
and communication with the server (code, abx, data values to send)

Similarly, the client needs to hold data for sliders, data lookups, meter calibrations,
timing control, the current radio, the list of radios for the selector etc etc

This data must be extracted at startup from the server based MYSQL database tables
and sent to data arrays on the client. (PHP arrays on the sever are converted to JSON format
for transmission to the client where they generate matching javascript arrays.)
For buttons and sliders, a data array entry is required for all 97 buttons and 29 sliders.
This is because unused buttons must still be represented in the array as *inactive* in order to
present them as inactive and greyed out to the user.
The server therefore builds full arrays of inactivated items and then selectively overwrites the active
items with configuration data from the database tables.
Eight tables are involved (not catcodes).
Three of these table have separate RS232, Icom CIV and Hamlib versions.

For **buttons,** the downloaded data array will be used in conjunction with a
fixed-coded button table. This has an entry for every button as below:

```
const BTN_075= 75;
const BTN_076= 76;
const BTN_SWAPAB = 79;
```

Every button has a number. First we define numeric constants for
these numbers. Most have the form: BTN_nnn. These are buttons
whose function and appearance can be user defined.
A few buttons with fixed internal functions have text identifiers,
eg: **`BTN_SWAPAB.`**

The fixed coded button table is of the form:

```
[BTN_075,    "btn075",    0],
[BTN_076,    "btn076",    0],
[BTN_VFOA,   "btnVFOA",   0],
[BTN_VFOB,   "btnVFOB",   0],
[BTN_SWAPAB, "btnSwapAB", 0],
```

Left column is **btnno** in database and the click reference.
Middle column is the button's **id** on the web page,
Right column stores the state of a toggling button.

For sliders**,** the downloaded data array will be used in conjunction with a
fixed-coded slider table. This has an entry for every slider as below:

```
[SLIDER_IF_WIDTH, "sliderIfWidth", "textIfWidth", "capIfWidth"],
[SLIDER_008,      "slider008",      "text008",     "cap008"],
[SLIDER_009,      "slider009",      "text009",     "cap009"],
```

`SLIDER_008` is IF-shift in some of my configurations. It has a non-specific numeric name
because it is NOT used as an identifier within the program code and so is free to be configured
for some other function if YOU wish.

Using **IF width** as the example:

- **`SLIDER_IF_WIDTH`** is a number constant (= 7) used to identify slider on position change etc and also is listed as **sliderno** in the database table.
- **`sliderIfWidth`** is the id of the slider - used to control it from code.
- **`textIfWidth`** is the id of the text data field to the right of the slider.
- **`capIfWidth`** is the id of the identifying caption to the left (if no on/off button)

Note that this fixed button and slider labelling in client is based on the use of the controls for my original FTdx101D development setup. It includes spares ie: **`SLIDER_SPARE_D.`**

In the database, the user can change the function and identification of sliders and buttons. The above labels do not change .... but they are internal to the code and so unseen.
I conveniently revert to this original setup for development purposes.

Once the web page has loaded, the control of the startup sequence is coded in **clientinit.js.**

The sequence is summarised below:

Ajax routines are mentioned - these are the link to the server (see: Ajax, MYSQL)

When piWebCAT is up and running, there are eight asynchronous processes running on timers. Their intervals are specified in the **timing** table. They are referred to in the sequence below.

Start up task sequence

- **catcontrol.php** (mainly HTML) is downloaded and rendered as the visual webpage.
  A large number of **javascript** files are imported from the server, including **jQuery**,
  some jQuery plugins and all the necessary G3VPX developed files.
  At this stage the visual page is populated with G3VPX original FTdx101D controls.
  The page includes the tpopup windows - initially hidden: log, memory pad and extra buttons.
- Empty data arrays are defined in client javascript for data from eight database tables
- Function ajaxGetMyRigArray() reads the single record **settings** table to get the
  name/identifier of the radio (eg: FTdx101D) This is the link to all the other tables.
- Function **`ajaxGetRadiosArray()`** is called to load the radios from the **rigs** database table.
  - this provides a list of radios for the drop down selector and also data for the current radio,
  ie **serial port** data (baudrate etc), **catcomms** (Icom CIV or ASCII)
  connection, ie whether RIG (direct to radio) or ENCAT (via EncoderCAT).
- VFOB display is hidden if vfobvis = 'N'.
- The serial port, connection and catcomms data is written to the page footer.
- **`ajaxSerialInit(...)`** initialises the RPi serial port with baudrate, stopbits, charbits and parity.
  (It also sets the serial port to RAW mode to prevent alteration of Icom CIV binary data)
- The remaining six data arrays are populated from the database (ie: buttons, sliders etc)
- **`setRadioOptions()`** inserts the list of radios into the drop down radio selector.
- **`setButtonCaptions()`** uses the buttons data array to replace ALL the buttons captions,
  set their colours and to set them active or inactive.
- **`setSliderCapions()`** sets slider active status and their labels (where no on/off buttons).
- **`getVfoAB()`** reads the current VFO selection from the radios
- The radio selector is set to display the current radio.
- Eight timing values are loaded from the **timings** array.
- The 10ms ajax queue read timer is started. (repetitive tasks are queued to avoid data collisions)
- The Tx meter buttons are initialised, Tx button selection is set to A (leftmost)
- A VFOA or VFOB click is emulated according to the current selection on the radio.
- The repetitive task timers are started for ongoing piWebCAT operation.
- **`initTunerA()`** stores the x/y coordinates of the blue mouse/thumbwheel tuner.
- **`initRxTuner()`** stores the x/y coordinates of the the tuning scale area.
  .

The final start up events are performed by the just started repetitive tasks.

- The frequency is read from the radio.
- The current band is derived form the frequency.
- The appropriate band tuning scale is installed and the read marker set.

Band and VFO change

Note that some of the above sequence is performed on band change.
(VFO change will often cause a band change)
In particular, the slider and buttons states are reloaded from the radio because many modern radios store a complete set for each band.
The buttons and sliders are unresponsive for a few seconds while the new settings are loaded from the radio. The client stores the most recent setting for each VFO so that on VFO swapping repeated reload from the radio is not needed. See: Dual VFO switching.

When operating split frequency mode between two bands, the control reload does not occur.
The frequency shows the transmit band frequency but the tuning scale stays on the receive band with the marker coloured olive at the band edge.

11.4 piWebCAT operation - the data request queue

The web browser is here referred to as the **client**. Raspberry Pi is **RPi**
piWebCAT communicates with the radios using the following data path:

Client (javascript) < ajax commands> **RPi** server (PHP code) <serial link> **radio**
< > MYSQL database

All data transactions are initiated from the client using **jQuery Ajax** requests (see Ajax, MYSQL)
This means that we cannot use Icom transceive (radio-initiated data transfers).

The PHP code on the server lives to do each single request and then ***dies.***

Client to server requests are:

- Timer driven repetitive tasks (eg: frequency change, meter updates)
- User driven tasks such as button and slider actions.
-

To avoid errors due to data collisions, all client > server tasks are managed in a **queue**.
The queue has six data types in 22 slots.
The slots are polled cyclically for requests bit in addition, certain data types are given **priority**.

The six data types are:

MOX	Radio Tx/Rx status. Responds to radio MOX and PTT. Read only.
METR	A meter read request (S meter or Tx meter) Read only, A and B slots.
FREQ	Frequency request. 4 slots, Read or write for VFO A or B
BUTN	Buttons - 4 slots. Read state or set state. Read or write for VFO A or B.
SLDR	Sliders - 4slots. Read state or set state. Read or write for VFO A or B.
DATA	Some other non-VFO specific requests

The queue is checked every 10ms (a database **timing** then deleted from the queue.
The next 10ms scan starts at the next point in the queue so that infrequent requests are not missed.
Frequency writes, other write items and meter reads are given extra high priority.

Priority data

From a priority viewpoint, data types are categorised as follows:

- **Frequency setting** A rapid sequence of frequency setting messages only occurs when the user is making a tuning action. (ie mouse wheel or drag tuning).
In addition, isolated frequency set messages occur from clicking the tuning scale or on band changing. When the repetitive queue scan process finds a frequency set message in the queue, it is given immediate priority.
This is appropriate because the user is totally focused on the rig frequency change response during this process.(Button and slider responses can be momentarily ignored because the user is unlikely to operate other controls exactly at the same time as tuning.
- **Buttons, sliders** Button and slider rig **setting** actions by the user give rise to isolated single messages. They can therefore be given high priority without risk of compromising the key background task of meter display and control synchronisation with the rig.
- **Meter updates** Updates of S meter read (and transmit metering reads) from the rig are the main ongoing background task needing a high priority. They occur all the time and so their maximum priority has to be limited to alternate message sending slots in order to allow the other background processes cocntrol syncing) to get a look in.
- **Syncing controls** This is the user-configurable synchronisation of selected on-screen buttons and sliders to their corresponding controls on the rig. They have the lowest priority and are managed cyclically to ensure that no control is repeatedly missed.

11.5 piWebCAT operation - MOX, S meter , Tx meters

MOX / Tx status.

The radio's transmit status is monitored every 1000ms (database **timer** table setting)
piWebCAT can be switched to transmit by the on screen MOX button
 - the MOX button lights up and the meter displays the currently selected Tx meter data..
However - we need to use piWebCAT with microphone PTT - hence the MOX data requests
to the radio to allow piWebCAT to respond to radio R/T status change.

For the FTdx101D, there is a read / write MOX data request: MX;
This responds to operation of the radio's MOX button, but not PTT.
I therefore configure the database MOX button command to use the TX; command
which does respond to PTT.

S meter and Tx Meters

There are five Tx meter buttons.

The **meter** table contains S meter and Tx meter records for each radio.
Some of its data is loaded to the client **MeterArray** at startup.
 ie: **mult** and **divide** for scaling and **usecal** (Y or N) for calibration table lookup.
Some is only used on the server for communication with the radio.

For my FTdx101D, the **meter** table has 9 records:
- S meter RxA and S meter RxB
- Seven Tx meters: PC, ALC, Comp, IDD, SWR, VDD and Temperature.

The five buttons are numbers **61** to **65** (code = TxmA to TxmE) These are fixed button numbers.
The allocation of five Tx values to these buttons is made in the database **meter** table.
(The unused values simply have btnno = 0;)
The left most button (TxmA) is selected at startup.

Operation

`smeterCAT()` is called by a repeat timer at an interval set in the **timing** database table.
A repeat interval of 100ms gives a satisfactory display.

The client knows the current VFO (for S meter) or the current Tx meter
and sends an appropriate read request via the ajax queuing system.

The returned data is optionally scaled and then adjusted by the calibration table
and then applied to the meter driver code.
At each update, the 250x110 pixel meter background is redrawn followed by the needle.
There is no flicker from the background redraw. (I do not have any slow PCs here !!)

S meter backgrounds
These are 250x110 pixel bitmaps.
piWebCAT holds six - one for S meter and one for each of the five Tx buttons.
They can be replaced for different displays.
If you design your own, then accuracy is achieved by the 20 point interpolated
calibration tables for each meter.

11.6 piWbCAT operation - Buttons

Testing. See: timing.disable

Startup

The state of most of the buttons is read for the current VFO (using the queue) at startup.
Exceptions to this are:
- The Tx Meter buttons whose control data is local to the client.
- The Band buttons (FTdx101D has no band-read, IC7000 uses band stacking resister.)

The database **buttons.action** field controls what happens when a button is clicked:

- action - **S** Single (momentary) - There is no client held state, so no startup action.
- action = **T** Toggled - start up state sets the buttons state.
- action = **G** Grouped (sharing a common code) The radio returns a selection code which allows the correct button to highlight (the rest of the group clear).
- action = **M** Tx meter button - no action as stated above
- action = **R** Slider reset button - sets slider to **def** value - local momentary action.
- action = **U** Unused button

Normal operation

(For band switching buttons - see the separate section on these)
When a button is clicked, the **action** field determines the outcome.
Button actions are write-only ... there is no feedback from the radio.

If button values are altered on the radio, the corresponding button states are only updated on piWebCAT if they are configured with **active=S** (sync) rather than **active =Y**.
There are up to 84 buttons to repetitively read. We limit the number of sync buttons because repetitive reading from the rig for all buttons would compromise the more important repetitive tasks (ie: meter read and frequency set)
However, piWebCAT provides a **Reconnect** button to resychronise piWebCAT's buttons and sliders after operation of controls on the radio. A further feature is that piWebCAT remembers this control data for the last band of each VFO so that simply toggling between VFOs doesn't need an auto-reload of control states.
The 27 **slider** controls are treated in the same way as buttons.

The button actions on clicking are:

- action = **S** Single (momentary) Send the programmed command.
  (I programmed these buttons to flash on for 250ms to give user feedback)
- action = **T** Toggled. Client code checks the button's on/off record and then toggles the button's appearance and sends the appropriate on or off command to the radio.
- action = **G** The button is in a group. Each button has a separate entry in the **buttons** table (or **buttonsciv** table) which contains the value to be sent with the command.
  The other members of the group are identified from the ButtonsArray on the client.
  (They have **action** = **G** and the *same* ***code***)
  The group has one or two records in the **catcodes** or **catcodesciv** table on the server
  These control the data transfers.
  Two records are present if there are separate VFO A and B settings.
  Code on the client highlights the clicked button and clears the rest of the group.
- action = **M** Tx meter button. The clicked buttons is highlighted and the associated record on the client is set so that the selected meter background and data will be displayed on subsequent transmit
- action = **R** Slider reset to default button. The ButtonData list in the client contains the id of the associated slider (rather than the button's id) The SliderArray (from the database at startup) contains the default value for reset buttons action.
  The slider is moved to the reset position and its numeric data display updated.

11.7 piWebCAT operation - Sliders

Testing. See: timing.disable

Startup

The state of the sliders is read for the current VFO (using the queue) at startup.

The slider position is set and the text value updated.

Normal operation

When a slider is moved, the new value is sent to the radio when the mouse or finger is lifted.
Slider actions are write-only ... there is no feedback from the server.

The **SliderArray in** the client was loaded at startup. Its data is used when the slider value is reset from the radio value at startup or changed by moving the slider.

When the slider is moved:
piWenCAT sends the appropriate message to the server.
On the server, the slider table in the database is interrogated to obtain information for a write of slider data to the radio.
For Yaesu there is then no response from the radio. From Icom there is an OK response.

When the slider value is changed by slider movement or a reset from the radio or a reset from the adjacent reset button, piWebCAT updates the numeric value text to the right of the slider.
Fields in the SliderArray control the data presentation of this numeric text:

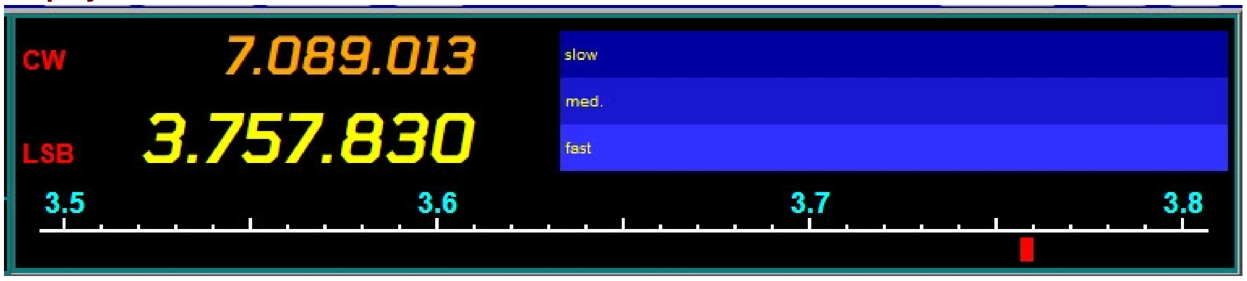
- **mult** and **divide** can scale the result.
- **decpoint** can insert a decimal point, eg: converting 3200Hz to 3.200kHz
- **offset** can be used to shift the value, eg 0 to 100 displayed as -50 to + 50
- **lookup** can invoke lookup table entries for non-linear cat > display relationships.
- **units** can be displayed after the value.

- additionally, **def** is the CAT value applied by the slider's reset button.

If slider related values are altered on the radio, the corresponding slider states are not updated on piWebCAT. There are 27 sliders to repetitively read. Such reading would compromise the more important repetitive tasks (ie: meter read and frequency set)
However, piWebCAT provides a **Reconnect** button to resychronise piWebCAT's buttons and sliders after operation of controls on the radio.

11.8 Frequency control - tuning

Display



There is a separate frequency scale for each band with a red frequency marker.
Clicking or touching the band can be used for coarse frequency positioning.

The blue tuning band has fast, medium and slow lanes.
Mouse drag or thumbwheel rotation over the tuning band will give a tuning rate of fast, medium or slow according to which lane is used.
Thumbwheel rotation elsewhere over piWebCAT will give the medium speed.
The fast. medium and slow rates for mouse drag and thumbwheel are user definable in the database.

Tuning

Tuning scale
The client code holds band limits and all the information to draw and calibrate the tuning scale for each band.   Driver code, using this data can:
- position the marker for a specified frequency
- yield a frequency when the scale is clicked

When this scale is clicked, the new frequency is placed in **freqSet** and a frequency write message is placed in the ajax queue for transmission to the radio.
The display is not updated by this process. (It happens later)

Tuning band
The web browser environment provides mouse related event routines to which I can add my own actions.
We have events: **mouse click**, **mouse down**, **mouse up**, **mouse in**, **mouse out**, **mouse move**.
All these are specific to the blue tuning area.
The code has continuous access to mouse position X and Y coordinates from anywhere on piWebCAT.
The tuning action is a horizontal drag.
We need the start position of our drag. I use mouse down to activate the process of drag recording.
I initialy used the X position at mouse down as my start X position. However, X is indeterminate with touch screens for finger down. This is because until you touch the screen, there isn't an X position!!
Fortunately, mouse move fires every few pixels and so I can use the X value at the first mouse move event as my X position origin. I store this in downX and store the frequency at this point as freqDown.

At subsequent mouse move events (until mouse up occurs), the displacement is X - downX.

The frequency F = freqDown + (X - downX)*pixelstep

where pixel step is Hz / pixel for the current lane (fast, medium or slow - database defined)

The new frequency, F is used to set the main frequency display and the marker on the tuning scale. **freqSet** is set to this frequency and also queued for transmission to the radio.

The process stops on mouse up or when the mouse coordinates are outside the tuning window.
I check this at every mouse move event.
(I cannot use mouse leave because it also fires when we change lane... which is unacceptable.
   - We need to be able freely change lane whilst drag tuning.)

Mouse move events may occur faster than they can be offloaded from the queue for send to the server.
This does not matter. There is one slot in the queue for FREQ write for VFO A.
If the next mouse move freqSet event arrives before the previous one in the queue has been transmitted, then it simply overwrites the previous one. We loose some fast mouse moves. We transmit tuning changes as fast as the system can handle them and we always transmit the latest available.

Mouse wheel
Mouse wheel tuning. There is a mousewheel event for each click of the wheel. This is translated into a frequency change according to fast, medium or slow lane position of the mouse pointer.
It is then handled as for mouse drag above. (You can use a bluetooth or OTG mouse with android)

Note that drag and mousewheel tuning band events queue the frequency change for immediate transmission to the radio.   This is essential to give a realistic fast tuning response.

By contrast, clicking on the tuning scale simply sets freqSet .. which is picked up and acted on a few 100ms later. I don't feel that this delay is a problem for a course change.

Use of the mouse wheel elsewhere on the piWebCAT window gives the medium tuning rate.

11.9 Band switching

piWbCAT development was done on two radios:

- **Yaesu FTdx101D** - a very new design
  A quick scan of other recent Yaesu models reveals use of the same control codes..
- **Icom IC7000** - a 15 year old design.
  A scan of modern Icom radios reveals the same command coding system but, the actual codes are different. ... Someone out there has a little work to do.
  The electronics of the CI-V interface on the radios appears not to have changed.

FTdx101D

It is hoped that the system described below is applicable to all other RS232 CAT radios. If there are issues, then the existence of a support group should help sort them.

The FTdx101D has **band select**, **band up** and **band down** CAT commands.
These are write-only. I could find no command to read the current band from the radio.

This is not a problem.
piWebCAT reads the frequency (main or sub) from the radio and then checks it against stored band limits. The repeat interval for such frequency read is set in the timing table. I use 200ms for main and 600ms for sub.
piWebCAT changes band if the main frequency enters a new band.
piWebCAT does not change band if frequency leaves the current band.... The red frequency marker simply changes to olive and stops at the band edge until the frequency returns into the band.

When piWebCAT does decide to change band:

- The tuning scale is redrawn for the new band.
- The band button selection highlight changes to the new band.
- Button and slider settings are reloaded for the new band.
  (on the FTdx101D, the roofer, mode, IPO/Amp etc change as expected
  but also most of the other buttons and sliders)

There can be a four second delay while this happens .... a good reason not to change band if we accidentally just drop outside band limits!!
On cross-band split frequency working, the reload does not occur on moving to the transmit band.
See also Dual VFO switching. piWEbCAT stores the latest control settings for VFO-specific controls in order to avoid load from radios on VFO swapping.

In summary:
The write-only band select command changes band on the radio.
The radio will switch to use its last used frequency on the new band.
piWebCAT will detect the frequency change in about 250ms and change band.

If band is changed on the radio, piWebCAT will detect the frequency change in about 250ms and change band.

Icom IC7000

I have a solution which works well and which is catered for by piWebCAT's standard CIV configuration options.
The only workable command that I could assign to a band button was a band-stacking register command. (Have I missed something here??!!)
The band stacking register holds three frequencies for each band:
- the most recently assigned, second most recent and third most recent.
For band button click, we configure in the database for piWebCAT to issue a write command to select the frequency of the 'most recent' register for the required band.

The command is command = 0x1A, subcommand = 0x01 data = 4 bcd digits (2 bytes)
eg: 0301 is used for 40m ...the 03 is 40m the 01 is the latest
So we configure **cmdtype** = C_S_DATA and datadigits = 4.

The band button sends the command to change frequency to the new band.
piWebCAT pick up the frequency change within 250ms and band change begins.

To the user, the response is no different to that on the Yaesu radio.

12.1 piWebCAT code - some basic information

The following pages use piWebCAT to give an idea of how this kind of web site works.
The web browser is referred to as the **client**. The Raspberry Pi as the **server**.

HTML code generates the web page.

Javascipt and jQuery code run on the client.
The code is in .js files or within tags <script> ...and.. </script> within web page code.

PHP code runs on the server and communicates with the radio via the RPi serial port.
It always resides between tags: <?php ..and ..?>

Style sheets (.css files) reside on the client and define **classes** for on-screen controls.
A **class** is then applied to a control (button, slider etc) to define its default appearance and behaviour)

Some basic statements:

- ***All client <> server communication episodes are initiated by the client.***
  The server code is programmed in PHP. PHP code on the server runs transiently for a job and then dies. It cannot be initiated by the serial port.
  (Icom 'CI-V transceive' radio initiated communications cannot therefore be supported)

- ***All client > server messages consist of a target .php filename on the server followed by parameters.***

- ***The target file parameters are always in the familiar form seen in URLs:***
  eg: `wcajaxserialinit.php?baudrate=38400&stopbits=1&charbits=8parity=0 &connection='RIG'&catcomms='RS232'`
- ***HTML code in the target file will be sent server > client to render (generate) web page content.***

- ***PHP code in the target file will run on the server.***
  Its scope is limited to the current request.
  It cannot directly share data with PHP code from other current or previous requests.

- ***The PHP echo command sends text back to the client.***
  - This may be data: piWebCAT uses **JSON** encoding to transfer complex data arrays from PHP code on the server to identical arrays in javascript code on the client.
  - PHP **echo** commands embedded in HTML text allow server code/data to influence client display content and also to place PHP generated javascript code on the client.

- ***PHP files may be referenced at the start of web page HTML code.***
  eg: piWebCAT has: `<?php  require_once("cat/phpfiles/wcpageswitch.php"); ?>`
  This PHP code runs at page starts to check if it is a restart requesting switch to another page.

- ***All javascript code sections in a web page behaves as one code area with sharing of data and code.***
  These are code sections loaded from a .js file, eg:
  ```
  <script src="cat/jqueryplugins/jquery.mousewheel.js"> </script>
  ```
  or code embedded within web page HTML, eg:
  ```
  <script>function onRadioChange(){setTimeout(ChangeRadio,1500);}</script>
  ```

12.2 HTML code

HTML files are **filename.htm** or **filename.html.**
If they contain embedded sections of PHP code, then they must be **filename.php**.

All website requests from browser to server are calls to download or run a file.
eg: the default piWebCAT RPi ip address is set to 192.168.1.117.
Placing this address in the URL bar of a web browser accesses port 80 on the RPI
and looks for a default web page in the web root folder. **index.html** loads as start page.
The START button in the index page then calls **catcontrol.php** which generates the main page.
catcontrol.php is mainly HTML code.

Web browsers have a rendering engine. On initial download, – this takes HTML code and interprets it into what you see visually. The lines of code are processed sequentially such that their sequence determines the arrangement of the final appearance.

HTML tables as an example of HTML coding:

piWebCAT makes extensive use of <table> components which have rows <tr> containing cells <td>.
I use tables within tables to achieve the required panels and positioning.
Some of the inner tables are not visible: I design with a border width of one pixel so as to be able to see the positions of the cells. Then when finished, I remove the borders.
The code below generates the S meter area and the Tx meter buttons below it.
A table is used for the top part of the page.
The table below is within that table on the left.

```
 <table class="noselect" cellspacing="2px">    <!-- inner table for meter and Tx buttons -->
  <tr>    <!-- first row  for S meter-->
    <td class="noselect" id="meterboxmain"  width="252px" height="96px" valign="middle"
bgcolor="black">
    </td>
  </tr>
  <tr>    <!-- second row for buttons -->
    <td class="noselect" height="44px" bgcolor="black" style="padding-left:3px">
       <span style="font-size:medium;font-weight:bold;color:aqua">TX</span>&nbsp;
button> <input name="btnTxmA" id="btnTxmA" type="button" class="buttontxmeter" style="width:34px"
              value="TxA" align="top" onclick="onPwcClick(BTN_TX_METER_A)" />
button><input name="btnTxmB" id="btnTxmB" type="button" class="buttontxmeter" style="width:36px"
               value="TxB" align="top"  onclick="onPwcClick(BTN_TX_METER_B)"  />
                           ..... 3 more buttons follow.....
    </td>
  </tr>
 </table>       <!--   end of inner table -->
```

In the above code:

- The S meter cell has **id="meterboxmain"**
  I can use this id to load the meter background images into this cell.
- The first button has **id= "btnTxmA"**
  I use this id to set the button's caption (from your database settings).
  I use **jQuery** code for this: **$("#btnTxmA").val('PO');** sets caption = 'PO' ie: Power Output.
- The first button has an on-click event javascript function: **onPwcClick(BTN_TX_METER_A**)
  onPwcClick(..) is used by all buttons. BTN_TX_METER_A is numeric identifier (= 61 - see buttons.js)
  This directs the javascript code in function onPwcClick(...) to carry out the correct button action.

12.3 PHP code

Note that the database editors use phpGrid, a complex product, purchased from HongKong.
phpGrid is discussed separately. The examples here are of other use of PHP code.

PHP code is always located between tags thus: <?php .... code here.... ?>

PHP code may be:

- **embedded in web page HTML code**
  - this will run when the web page is loaded (rendered).

- **in a dedicated .php file on the server**
  - this will run when the file is launched from the client.

Embedded PHP code.

An example:
It is the version statement at the left of the footer bar. (on all three webpages)
The version is in **wcversion.php**, a very simple file as shown:

```
<?php
   $ver = 1.001;
   $version = "1.001";
?>
```

At the start of the web page we have: `require_once "cat/phpfiles/wcversion.php";//`

The file sets PHP variables $ver and $version (All PHP variables must begin with a $ character)

Embedded PHP code is only valid for the duration of the initial rendering of the page.
So these definitions are lost once page rendering is complete.

Further down the web page HTML code, we write the text of the footer:

```
<span style="color:yellow">piWebCat version <?php echo $version; ?></span>
```

`<span>....text ... </span>` is a way of formatting text and labelling text
- We set attributes in <span>. which are applied to the enclosed text.

A <span> tag can have a number of attributes, one of which is style.
Style then has a wide range of attributes ... as used in .css style sheet files.
Here we simply set text color to yellow. (We are already working on blue background)

The first part of the text is: `piWebCat version`

The second part is embedded PHP code: `<?php echo $version; ?>`

The extensively used PHP **echo** command is run on the server at this point in building
the HTML text to send to the client.
The echo command sends text to the client in the middle of rendering the web page,
and so the value of $version is inserted in the web page..

So we see on web page footer : version 1.001 .

Embedded PHP code example 2

The database editor uses **phpGrid** to show the database tables in a grid presentation.
Twelve grids are built (from 12 database tables) when the page is constructed on the client.
... at any one time we hide 11 of them!

phpGrid is purchased from a Hongkong company. I have an OEM distribution license.
phpGrid allows the creation of complex database aware grids with only a few lines of PHP code.
It has in line editing on grid.

A single line of PHPcode can set one of the grid's columns to behave as a drop down list selector.
For example:
We have a list of radios in the database **rigs** table.
We need to present the radios in a drop down list for selection.

To do this, we can add a PHP code line to the grid initialisation .. of the form:

```
$bg->set_col_edittype("active","select",FT2000:FT2000;FTdx101D:FTdx101D;....,false);
```

The list controlling string here is: **`FT2000:FT2000;FTdx101D:FTdx101D;....`**
`FT2000:FT2000` is a key - data pair.
- The first **`FT2000`** is the identifier and is written to the database table
- The second **`FT2000`** appears in the drop list for selection

(They don't have to be the same ... but they are the same in all piWebCAT usage.)

Therefore, to generate a drop list of radios, we need a string:
`FT2000:FT2000;FTdx101D:FTdx101D;IC7000:IC7000;` **`etc`** for all the radios.

File **`wcradioselector.php`** generates this string from the list of radios in the database.
It places the radiolist string in variable **`$RadioSelect`**.

The editor page is **`catedit.php.`**

We place: **`require_once "cat/phpfiles/wcradioselector.php"`** at the start of **`catedit.php`**.

This runs the code in **`wcradioselector.php and`** results in **`$RadioSelect`** being available
on the server *for the duration of the page build*.

We add **`set_col_edittype(..)`** to the set up of each grid, using the radio list in **`$RadioSelect`**:

```
  $bg -> set_col_edittype("rig", "select",$RadioSelect,false);
```

This makes the rig column, on edit present drop down radios list.

12.4 PHP code files - initialising the RPi serial or USB port

Client javascipt code sends a php file name request to the server to do this specific task.
The file is `wcajaxserialinit.php.` Parameters are added after the filename.
An **Ajax** command is used to generate the call. In this call, there is no data to return to the client.
AJAX is demonstrated elsewhere using a more complex situation that returns data to the client

```
 wcajaxserialinit.php?baudrate=38400&stopbits=1&charbits=&parity=0&connection='USB'
                &catcomms='HAMLIB'&rigfix='nofix'&hamlib=1040&civaddr=0,&vfomode='Y'
```

wcajaxserialinit,php uses **PhpSerial.php** which has **class**: $serial containing the functions
used to initialise and access the RPi serial or USB port connected to the radio.
These functions must be accessed as $serial->deviceSet (..) etc etc.
This is because the method (subroutine) deviceSet(..) is defined within the class $serial.

```
<?php
  ini_set("output_buffering","On");  error_reporting(E_ALL);
  ini_set('display_errors', '1'); ob_start();  session_start();

  require_once 'PhpSerial.php';   require_once 'wchamlib.php';

  $baudrate = $_POST['baudrate'];
  $stopbits = $_POST['stopbits'];
  $charbits = $_POST['charbits'];
  $parity = $_POST['parity'];
  $connection = $_POST['connection'];
  $catcomms = $_POST['catcomms'];
  $rigfix = $_POST['rigfix'];
  $hamlib = $_POST['hamlib'];
  $civaddr = $_POST['civaddr'];
  $vfomode = $_POST['vfomode'];

  if($connection == 'USB'){$tty = "/dev/ttyUSB0";} else {$tty = "/dev/ttyAMA0";};

  $_SESSION['tty'] = $tty;              $_SESSION['connection'] = $connection;
  $_SESSION['catcomms'] = $catcomms;    $_SESSION['rigfix'] = $rigfix;
  $_SESSION['hamlib'] = $hamlib;        $_SESSION['civaddr'] = $civaddr;
  $_SESSION['vfomode'] = $vfomode;

  if($catcomms == 'HAMLIB')      // For Hamlib we communicate with the API,
  {                              // not directly with the rig.
    $ret = $Hamlib->startRigctld($hamlib, $tty, $baudrate,
                                 $stopbits, $charbits, $civaddr, $vfomode);
  }
  else
  {
    $serial->deviceSet($tty);           $serial->confBaudRate($baudrate);
    $serial->confParity($parity);       $serial->confCharacterLength($charbits);
    $serial->confStopBits($stopbits);   $serial->confFlowControl("none");
    $serial->deviceSetParams();         $ret = "OK";
  }
  echo $ret;
?>
```

Note the separate startup call for Hamlib which is in file wchamlib.php.
We start **rigctld** which is the Hamlib API. **$hamib** is the rig number, **$civaddr** is only for Icom.
piWebCAT communicates with the rigctld socket (localhost:4532). rigctld communicates with the rig.

The ten values are picked up from the calling parameter string using $_POST['baudrate'] etc

`$serial->deviceSet(..)` tells the PhpSerial.php code to use RPi serial port: /dev/ttyAMA0
or the USB connection: /dev/ttyUSB0.
(Note that an access permission rule must be set for /dev/ttyUSB0 -   See; 14.4 SD card config)
The four serial settings are then applied, ie: baudrate, parity, bits/char and stopbits.
Although this PHP process and its data ***die*** when the job is done, the serial port in the RPi
will retain its settings for ttyAMA0 or ttyUSB0.
So when we subsequently use the port, we only need to call **`deviceSet(/dev/tty....)`**
to use the device. (It does not need reconfiguring)

The **`$_SESSION`** commands set up ***session variables*** which are retained by the webserver and CAN be accessed by subsequent PHP processes:

$connection is **SERIAL** or **USB** (or **ENCAT** - via my EncoderCat unit… not with Hamlib)

$catcomms is

- for piWebCAT direct control: **ASCII** (text based Yaesu, Kenwood),
  YAESU5 (older 5 byte Yaesu)
  CIV (for Icom)
- or **HAMLIB** (using Hamlib rig database and API)

12.5 Javascript code and jQuery

Javascript code runs on the client.

Its locations are:

- **In a server file**: filename.js file. piWebCAT javascript files are in folder cat/js.
  eg: at the start of catcontrol.php, javascript loads include:
  ```
  <script src="/cat/js/clientinit.js"></script>
  <script src="/cat/js/meter.js"></script>
  <script src="/cat/js/tuner.js"></script>
  ```

- **Embedded in an HTML web page file**
  eg: - towards the end of catcontrol.php we have:
  `<script> $(window).on('load', function() { pwcInit();}) </script>`
  This calls function `pcwInit()` when web page loading is complete.
  `pcwInit()` (in file clientInit.js) calls several initialisation functions for the web page.
  It cannot run until page build is complete because, for example, it cannot set captions and colors on buttons that don't yet exist.
  (hence the use of `$(window).on('load'..` to delay `pcwInit()` until load is finished)

All Javascript, embedded and file-based becomes part of one composite javascript code block for the lifetime of the webpage. Any function defined in embedded code or in a loaded file is available for use by any other function.

Javascript can read and set the properties of controls on the webpage (eg: buttons and sliders)
JQuery make this much easier:

jQuery

jQuery extends the javascript language with an extensive set of commands that are easier to use.

jQuery commands can all begin with **jQuery(.....** but **$(....** is shorter and therefore used.

Example:

Every component on the web page is assigned an **id** field, eg:

```
<input id="btnTxmA" type="button" class="buttontxmeter" value="TxA"  ..etc
```

This is the leftmost Tx meter button on my FTdx101D. Its **id** field is "**btnTxmA**".
(Actually a very long definition ..so I truncated it to fit it on this page!)
The button definition above has `value = "TxA"`. For HTML buttons, this sets the caption.

To change the caption to "**PO**" with jQuery, we simply use `$("#btnTxmA").val("PO");`

I cannot remember all the wide range of jQuery functions.
I have downloaded and printed the jQuery manual ... but I don't use it.
I just Google what I want to know..... Every search yields multiple helpful answers.

Note the `class="buttontxmeter".`
`buttonxmeter` is a button design specification define in file **styles/buttons.css**

12.6 jQuery Ajax MySQL access

The piWebCAT control web page has 67 + 24 extra buttons and 27 sliders
.
Configuration information for these controls is held in the MySQL **radio** database on the server.

Some of this configuration information is needed on the client, some is needed on the server.

Sliders

For **sliders** there is a single database table: **sliders**
(or **slidersciv** for Icom CIV or **slidershl** for Hamlib)

- Fields that control slider **appearance and operation** need to be accessible on the **client**.
  eg: rig, caption, sliderno, active, code, abx, mode, min, max, def, mult, divide, offset
  units, lookup, decpoint,
- Fields that control **communication with the radio** need to be on the **server**
  ie: code, abx, readbytes, setbytes, answerbytes, readmask, setmask, answermask

Fields **code** and **abx** are the operational link between client slider data and server slider data.

Buttons

For buttons, client data is the **buttons** table and server **data** in the **catcodes** table.
(and **buttonsciv** or **catcodesciv** tables for Icom CIV radios or **catcodeshl** for Hamlib)

Loading to the client at startup

At piWebCAT startup, we need to load button and slider data to array variables in the client.

In fact we need to do it also for tables: **lookups**, **meter**, **timings**, **radios** and **settings**.

The following description describes how this is done using **sliders** as the example.

12.7 Ajax and MySQL - loading slider data to the client

I will use **sliders** to illustrate the operation of Ajax client <> server data transfer
and MySQL (MariaDB) database access.

At piWebCAT startup, this process extracts 16 fields for 27 sliders in MySQL table **sliders**
and places the data in a data array variable on the client.
Javascript code on the client can thereafter use this data array to:

- Set slider captions, ranges, and the presentation of the numeric slider value.
- Control communication with the server.

Note that the captions are only needed for those sliders that do not have an associated button.
(If there is a button to the left, then the caption should have been set to 'nocap'.)

In clientinit.js we have the following code to get the slider data from the server:

```
var SliderArray = [];    // set up an empty array to receive the data

function ajaxGetSliderArray()
{
  var urlparams = 'rig=' + rig + '&job=' + GET_SLIDERS;
  $.ajax({
   type: 'POST',
   url: 'cat/phpfiles/wcajaxinit.php',
   data: urlparams,
   dataType: 'text',
   cache: false,
   async: false,
   timeout: 5000,
   success: function(data){SliderArray = $.parseJSON(data);},
   error: function(data){console.log('error');}
  });
}
```

`function ajxGetSliderArray()` is one of a number of similar functions called at startup

This is standard Ajax format and is the ONLY code needed on the client to do
the job of getting the slider data into **`SliderArray.`**
It is a beautifully simple way of controlling a quite complex task.
Let us use the FTdx101D as example:

First: note that variable **`urlparams`** `is` set to: **`'rig=FTdx101D&job=2'`**

`GET_SLIDERS` is a numeric constant set to a value of 2 in **enums.js**

A **`GET_SLIDERS constant = 2`** is similarly defined in wcenums.php on the server.

Ajax will send to the server: **`url + '?' + data`** which for this job is:

```
        cat/phpfiles/wcajaxinit.php?rig=FTdx101D&job=2
```

Transmission options are standard internet POST or GET ... we use **POST**

We shall return to the other Ajax fields later looking at processing the result.

On the server - selecting radio and job

File: **`wcajaxinit.php`** is used to create data arrays from eight database tables.
We are just looking at the **sliders** table as an example
(or **sliderściv** for Icom CI-V direct or **slidershl** for communication via Hamlib)

`wcajaxinit.php` does this slider job an then ***dies***.

At its start we have to build in two more .php files.

```
   require_once 'wcmysql.php';
   require_once 'wcenums.php';
```

`wcmysql.php` contains database access info.: **`$DbServer, $DbUser, $DbPw, $Database`**
`wcenums.php;` defines numeric constants. In our example **`GET_SLIDERS = 2`**

The code is then:

```
$an = new ajaxinit;          // create an instance of the ajaxinit class
$an->JobList();               // run the JobList() method  (function)

class ajaxinit                 //  all functions in this file are in class ajaxinit
{
  public function JobList()
  {
    $job = intval($_POST['job']); // get the job code from the URL
    $rig = $_POST['rig'];         // get the radio from the URL
    switch($job)
    {
      case GET_BUTTONS:
        $this->GetButtonData($rig);
        break;
      case GET_SLIDERS:               // our job is GET_SLIDERS
        $this->GetSliderData($rig);    //  do the job
        break;
      case GET_LOOKUPS:
         .... etc ...
```

The code above extracts **`$job`** and **`$rig`** from the calling URL POST data and then directs to function: **`$this->GetSliderData($rig);`**
`$this->` because the function is in ***this*** **`ajaxinit`** class.
`$rig` is a parameter passed to the function to inform it of which radio to get the sliders for.

See also Inspect element feature of web browsers.

On the server - extract the data from the database

`function GetSliderData()` extracts selected fields from the database **sliders** table
for all the sliders for FTdx101D and builds them into a PHP data array.
(or **slidersciv** for Icom CI-V direct or **slidershl** for conenction via Hamlib)

I have split the function into two halves for discussion purposes:

```
public function GetSliderData($rig)
{
  global $DbServer;      //  The global declarations are always needed
  global $DbUser;        //   to bring external variables into PHP functions.
  global $DbPw;
  global $Database;
  global $catcomms;

      // choose the correct table -  Icom CIV or RS232 ?  (catcomms is from a SESSION variable)
   if($catcomms == 'CIV') {$table = 'slidersciv';} else {$table = 'sliders';};

     // define an empty array to receive the data
   $SliderArray = array();

     // connect to the database using the data from wcmysql.php
   $conn = mysqli_connect($DbServer,$DbUser,$DbPw,$Database);
   if (mysqli_connect_errno($conn)){echo "failed to connect to MySQL: "
              . mysqli_connect_error(); }

    // set up a database query  .. in this example
      SELECT sliderno, active, caption ..etc ..... FROM sliders WHERE rig = 'FTdx101D';

   $sql = "SELECT sliderno, active, caption, code, abx, mode, min, max, def,
   mult, divide, offset, units, lookup, decpoint ";
   $sql .= "FROM " . $table . " WHERE rig = '" . $rig . "';";  // . is concat

       // Run the query to place data extraction information  in variable: $result
   $result = mysqli_query($conn, $sql);
```

Using **$result** we can extract the requested database data, one record at a time.
For each retrieved record, the fields are returned as **`$row[0]`**, **`$row[1]`** ..etc
in the order that we made in the requesting query.

```
   $i = 0;
   while($row = mysqli_fetch_array($result)) // keep repeating until false
   {
     $SliderArray [$i++] =   // $i++   means use the value of $i and then
      array(                       // increment it ready for the next record
        'sliderno' => $row[0],     // sliderno was the first field in our query
        'active' => $row[1],        // active was the second field in our query
        'caption' => $row[2],
        'code' => $row[3],
        'abx' => $row[4],
        'min' => $row[5],
        'max' => $row[6],
        'def' => $row[7],
        'mult' => $row[8],
        'divide' => $row[9],
        'offset' => $row[10],
        'units' => $row[11],
        'lookup' => $row[12],
        'decpoint' => $row[13],
        'aval' => 0,              // used to store VFOA value on VFO switching
        'bval' => 0,                        // used to store VFOA value on VFO switching
        'xval' => 0               // unused

        );
    }; // go back no for the next record
    mysqli_close($conn);

   echo json_encode($SliderArray);
 }
```

The outcome of the above code is a PHP array with elements:

```
      $SliderArray[0]['sliderno']    //  the 15 fields of the first slider
      $SliderArray[0]['active']
       etc...
      $SliderArray[1]['sliderno']    // the 15 fields of the second slider
      $SliderArray[1]['active']
        etc ..
```

Now the magic bit..

`json_encode($SliderArray);` generates a copy of the whole slider array in JSON format

`echo json_encode($SliderArray);` generates the JSON code and sends it back to the client.

Then, in client javascript, the received JSON data, is built into an identical javascript array
by simply using:

```
       SliderArray = $.parseJSON(data);
```

Thus, single line of PHP code on the server encodes and sends the whole sliders array to the client and then a single line of code on the client generates a matching javascript array.

Data returned to the client

The JSON encoded data is transferred as a very long character string as below:

```
[{"sliderno":"1";"active":"Y";"code:"AMCO";..etc....},{"sliderno":"2";. etc
```

The originating Ajax code is restated below:

```
var SliderArray = [];    // set up an empty array to receive the data

// get slider data  array from server
function ajaxGetSliderArray()
{
  var urlparams = 'rig=' + rig + '&job=' + GET_SLIDERS;
  $.ajax({
   type: 'POST',
   url: 'cat/phpfiles/wcajaxinit.php',
   data: urlparams,
   dataType: 'text',
   cache: false,
   async: false,
   timeout: 5000,
   success: function(data){SliderArray = $.parseJSON(data);},
   error: function(data){console.log('error');}
  });
}
```

On a successful transfer, the jQuery code is: **`SliderArray = $.parseJSON(data);`**

Other data arrays

We build seven other arrays at startup:
 ButtonArray, LookupArray, TimingArray, MeterArray, MeterCalArray, RadiosArray and MyRigArray.
These remain on the client for the duration of the session.
They are used at start up for setting button captions and colours.
They are used at startup, VFO change, band change for data formatting (eg: mult, divide, decpoint)

12.8 Client server communications after startup

After startup, data transfers use **ajaxdata.js** on the client accessing **wcajaxdata.php** on the server

All transfers are directed through function:

```
        sendWebcatMessage(param, rxab, task, code, jobdata, opasync)
```

This uses an Ajax call similar to the sliders example above.

There are a number of ongoing, repetitive, asynchronous tasks that must use this data route.
Their individual repetition periods are specified in milliseconds in the **timing** table.

They are:

- **getMoxStatus** - We need to switch piWebCAT to transmit mode if microphone PPT is pressed.
- **smeterCAT** -- Read the meter (Smeter or a transmit meter with a repitition rate high enough for an effective display).
- **getFreqMainCat** - piwebCAT needs to respond to tuning actions on the radio - but this is not its normal operation and so 200mS is satisfactory.
- **getFreqSubCat** - less important - I use 500mS
- **checkBandMain** - monitor the current frequency and change the band display etc if necessary. (band change will trigger reading of mode, DNR setting, roofer selection etc etc) 1000mS is adequate
- **checkModeMain** - check for a radio initiated mode change (eg: 2000ms)
- **checkSetFreqMain** - Mouse drag and thumbwheel tuning actions change the variable: **freqSet** This task examines freqSet every 50ms and sends changes to the radio.

Note that piWebCAT does not read the current band from the radio.
It reads the frequency and checks it against band limits.

For example: I do not have a 4m band radio. But, for testing purposes, when I wind the frequency on the IC7000 upwards from 50 Mhz, piWebCAT's tuning display flips to the 4m tuner when I hit 70 Mhz.

In addition to the repetitive traffic described above, **button** and **slider** messages to the server occur at random times.

A javascript **queuing system** is therefore used in **ajaxdata.js** on the client *to* ***avoid data collisions***.
The queue has one read and one write slot for each of:
MOX, meter A and B, frequency A and B, Button A and B, slider A and B and data X.
A new incoming message will replace a waiting unsent message
The queue is interrogated every 10ms in a circular fashion for messages to send..

This queuing system ensures that all messages are dealt with without data collisions.

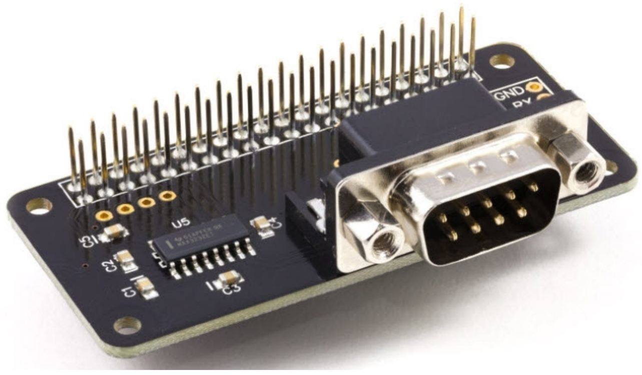
13.1 Serial Pi Zero and G3VPX piWeb CAT PCB - A RS232 and CI-V interface for the RPi.

(Not needed with USB connection to radio)

Serial Pi Zero

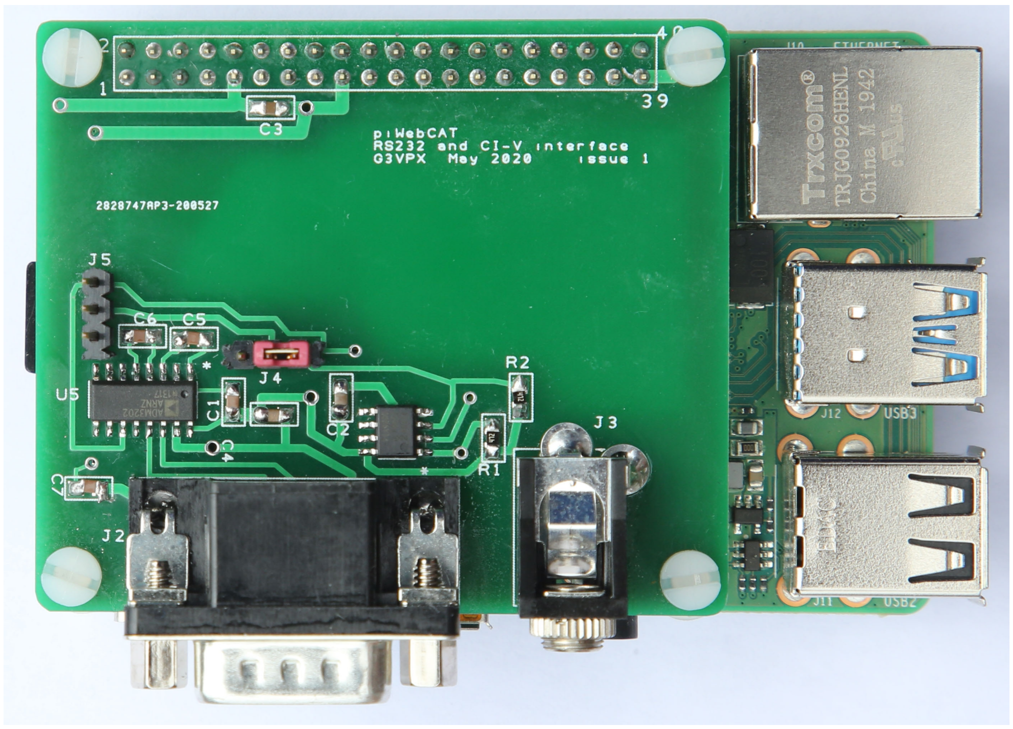
For RS232 CAT connections the easiest option is to purchase a Serial Pi Zero board.
This is the size of the small Pi Zero computer but fit any RPi GPio connection.
It uses the serial port connections on GPIO pins 8 and 10 to through an RS232 driver device. The cost at the time of my purchase was £8.49.

The RPi 3 has two serial ports, one of which is high quality and with high baudrate capability.
My initial serial port development was with RPi3 computers on a commercial project.
The fast serial port is switched to GPIO pins 8 and 10.
The RPi4 has five fast serial ports . but I chose not to use the four new additions.
Reason: (i) The RPi 3 arrangement is just as effective (ii) the Serial Pi zero uses the port on pins 8 and 10 !! (iii) The information on RPi 4 ports became available some time after the RPi 4.



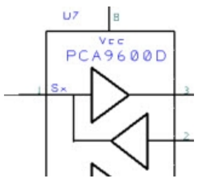
G3VPX piWebCAT PCB

This is a commercially manufactured PCB with drivers and interface devices for both RS232 and Icom CI_V (3.5 mm mono jack socket). J4 - RS232 left CI-V right (labeling missed!)
It has been made the full width of the RPi to bring the connectors to the board edge and thereby to the side of an enclosing box. I have left a large free area on the board (with no ground plane) so that it could be used for 'breadboarding' of new circuit ideas.



G3VPX piWEbCAT PCB

The RS232 driver is a **ADM3202** device (as used by Yaesu in the FTdx101D)



The CI-V driver is one half of a NXP PCA9600 device.
Its intended use is as a dual I2C line driver for I2C communication via standard CAT5 LAN cable. It separates each bidirectional I2C line (ie; SCL and SDA) into two separate lines - one Rx and one Tx - each to a CAT5 pair.
I use it in reverse to combine GPIO serial Rx and Tx lines into a single CI_V line.

Both buffers of the pairs have open drain outputs and so need pull up resistors.
The Tx buffer is the lower one in the above image. Its 'logic low' pull down voltage is 0.76V max. which is not low enough to trigger as logic low by the Rx buffer. This means that the outgoing negative going pulses do not simultaneously come back on the Rx line.
This 0.76v low logic level works fine with my IC7000.
Examination of the IC7000 interface circuit confirms that it should be ok.
I checked the CI_V interface of a current Icom radio and it is almost identical to that on the 15yr old IC7000.

This device removes the problem of outgoing Tx pulses appearing at the Rx input on the processor.
Therefore the RPi serial does not have to deal with such unwanted input .... and so doesn't do so.

This concurrent return of transmitted serial data is referred to as CI-V echo.
Hamlib rigctl operates with or without echo and so there is no problem.

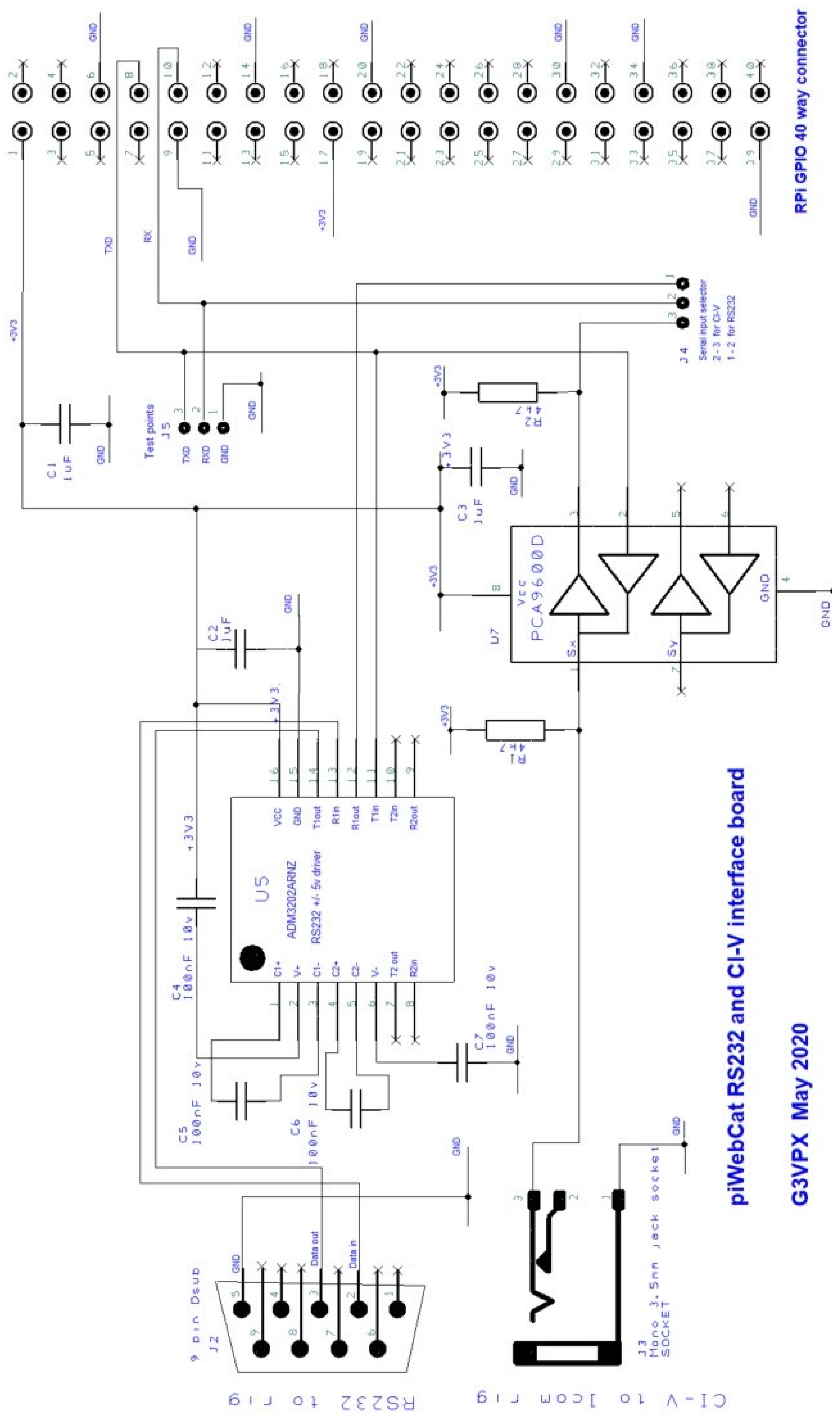
piWebCAT without Hamlib does not tolerate the echo.
This is not a problem using the piWebCAT PCB.
However it would be a problem using a Serial piZero RS232 card with the now obsolete RS232 <> CI_V interface from Icom.
It is not an issue with USB connection.
(I have not run piWebCAT with Icom USB. I don't anticipate a problem ... feedback please!)

I can supply the bare piWebCAT PCB for £4 including P&P.

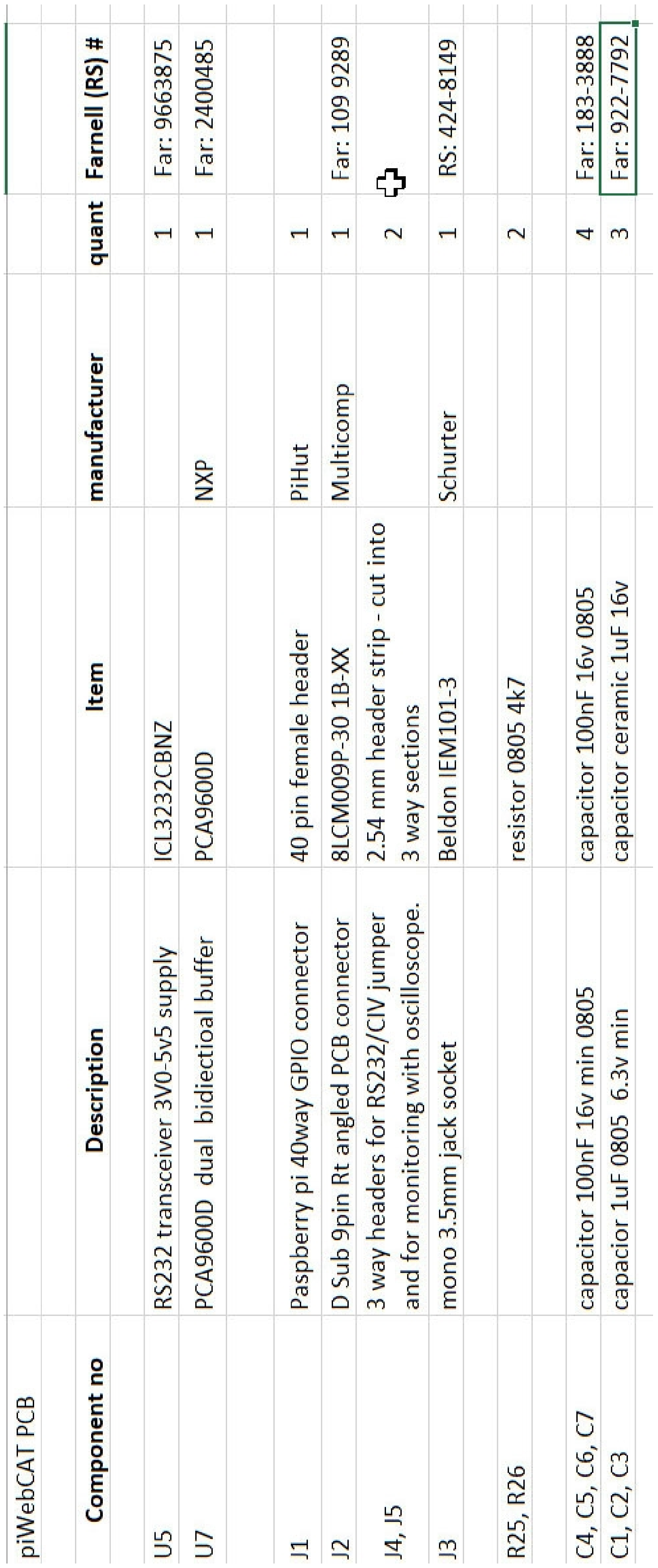
See Section 14.2 Support. Supply of SD card &PCB

(An initial order of 100 PCBs from JFCPCB, China arrived 8th June 2020)

13.2 piWebCAT PCB schematic


RPi GPIO 40 way connector
TXD
RX
GND
+3V3
C1 1uF
Test points J5
TXD RXD GND
J4 Serial input selector 2-3 for CI-V 1-2 for RS232
R2 4k7
R1 4k7
C3 1uF
U7 PCA9600D
Vcc
GND
Sx
Sy
C2 1uF
U5 ADM3202ARNZ RS232 +/- 5v driver
VCC GND T1out R1in R1out T1in T2in R2out
C1+ V+ C1- C2+ C2- V- T2 out R2in
C4 100nF 10v
C5 100nF 10v
C6 100nF 10v
C7 100nF 10v
J2 9 pin Dsub
Data out
Data in
J3 Mono 3.5mm jack socket
RS232 to rig
CI-V to Icom rig
piWebCat RS232 and CI-V interface board
G3VPX May 2020

13.3 piWebCAT PCB component list



piWebCAT PCB

Component no	Description	Item	manufacturer	quant	Farnell (RS) #
U5	RS232 transceiver 3V0-5v5 supply	ICL3232CBNZ		1	Far: 9663875
U7	PCA9600D dual bidiectioal buffer	PCA9600D	NXP	1	Far: 2400485
J1	Paspberry pi 40way GPIO connector	40 pin female header	PiHut	1	
J2	D Sub 9pin Rt angled PCB connector	8LCM009P-30 1B-XX	Multicomp	1	Far: 109 9289
J4, J5	3 way headers for RS232/CIV jumper and for monitoring with oscilloscope.	2.54 mm header strip - cut into 3 way sections		2	
J3	mono 3.5mm jack socket	Beldon IEM101-3	Schurter	1	RS: 424-8149
R25, R26		resistor 0805 4k7		2	
C4, C5, C6, C7	capacitor 100nF 16v min 0805	capacitor 100nF 16v 0805		4	Far: 183-3888
C1, C2, C3	capacior 1uF 0805 6.3v min	capacitor ceramic 1uF 16v		3	Far: 922-7792

14.1 piWebCAT file downloads

piWebCAT.pdf http://piwebcat.g3vpx.net/files/piWebCAT.pdf
A 222 page PDF of this website

Startup guide http://piwebcat.g3vpx.net/files/piWebCAT_startup_guide.pdf

Summary pdf piWebCAT 9 page summary (PDF)

piWebCAT.zip http://piwebcat.g3vpx.net/files/piWebCAT.zip
The complete web site structure (locate in /var/www/html on the RPi)
This includes the help system which is a copy of this website.

radios.sql.zip http://piwebcat.g3vpx.net/files/radios.sql.zip
G3VPX database backup containing:
FTdX101D, IC7000, FT847, FT920 - piWebCAT configurations
FTDx101D-H, IC7000-H Hamlib /piWebCAT configurations.
Empty Station log table.
(Caution - it will replace your database)

LibrarySQL.zip http://piwebcat.g3vpx.net/files/LibrarySQL.zip
A useful library of useful SQL scripts for database manipulation.

MYSQL Front http://mysql-front.freedownloadscenter.com/windows/free/
- free download Database browser /editor

HeidiSQL www.heidisql.com - free download Database browser / editor

Hamlib - rigctl and rigctld See Adding Hamlib to an SD card

https://github.com/Hamlib/Hamlib https://github.com/Hamlib/Hamlib/wiki/Documentation

RPi Micro SD card
This is a 16GByte micro SD card image for the Raspberry Pi.
It is available by emailing piwebcat@g3vpx.net. (SD card or 3.5Gbyte zipped image on a DVD)

This is fully configured with RPi OS, MariaDB, PHP. phpMyAdmin, Apache2 web server, pure-ftpd FTP.
It has the 'radio' database configured radios as above in radios.sql.zip
It has the piWebCAT website installed. (This includes a copy of this website as a built in help system.)
It has HAMLIB rigctl and rigctld installed.
It has a Mumble VOIP audio client and server installed.
It has IP address 192.168.1.117

Raspberry Pi imager - use for loading **.img** image files onto a (fast !) SD card.
It does not appear to read a card into a **.img** file. Use **WIn32DiskImager** for this.

Raspberry Pi Imager for Windows For **Choose OS** select **Use custom** and then select your .img file)
Raspberry Pi Imager for macOS

For a **low cost multi micro SD card copier**, see section 1.8 Raspbery Pi preloaded SD card

14.2 piWebCAT - support - supply of SD card and PCB

Configurations on RPi micro SD card

piWebCAT direct control has been developed using only four radios:

• Yaesu FTdx101D	ASCII text configuration (as used in modern Yaesu, Kenwood, Elecraft)
• Icom IC7000	CIV - Icom CI-V configuration
• Yaesu FT847	YAESU5 configuration - Older Yaesu '5 byte' radios.
• Yaesu FT920	YAESU5 configuration - Older Yaesu '5 byte' radios.

The can be used as templates for control of other similar radios.

piWebCAT via Hamlib rigctl has been configured for:

- Yaesu FTdx10D
- Icom IC7000
- Generic transceivers: Transceiver-H-A, Transceiver-H-B and Transceiver-H-C
  These are an evolving progressive build from A to C used in the learning guide.
  They provide a template for Hamlib control of any one of Hamlib's 250 supported radios.

Support group

We need a support /messaging group to deal with any problems arising from
operational issues, radio compatibility and development ideas.
Also, using MySQL Front, it is possible to share configuration data.

I have set up a support group at **piwebcat@groups.io**

Micro SD card, PCB or PCB kit available from G3VPX:

piWebCAT bare PCB @ £4.00 including p&p
piWebCAT PCB kit of parts + PCB @£15.00 including p&p

Preconfigured 16Gbyte fast class10 micro SD card @ £7.50 including p&p
Zipped SD card image on DVD (Created with Win32 Disk Imager) @ £4.00 including p&p

Please email using piwebcat@g3vpx.net (This NOT my PayPal email !)

I have not at this time provided a zipped SD card image download.
This would be 3.5Gbyte which is too large for my well populated 5Gbyte web hosting.
In the event of very high demand, then I would consider increasing the web space
to 10 Gbyte and hosting a downloadable image.

14.3 piWebCAT - development tools

The following were essential in achieving successful development of this project:

My Siglent SDS2202X-E oscilloscope.

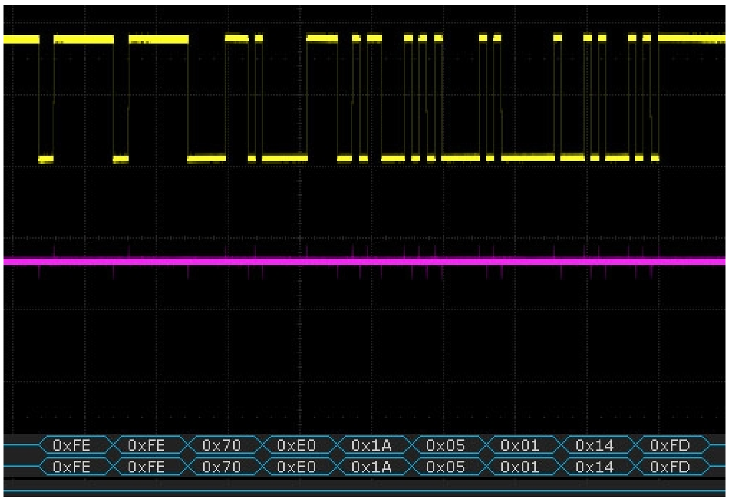
This is a superb 200 MHz 2 - channel storage scope from China.

Its vital feature in developing piWebCAT was its very flexible ability to capture
and expand displays of serial data and to decode the data as ASCII or Hex.
(Note that piWebCAT was developed using serial radio connections.
USB connectivity was added at the end of development, after data problems were solved)

This allowed me to examine in detail the data streams to and from the RPi serial port.

One particular issue was in reading the band stacking register on my Icom IC7000.
The radio was observed to return a 26 byte data string.
I could find no documentation on this at all - I had no idea that there were 26 bytes to receive.
Firstly, it is essential to know how many bytes will be in a response.
- Without this scope, I could not determine this.
Secondly I needed to know the content / format of the string.
The decode on the scope revealed BCD coded frequency and operating mode at the
start of the data section. That was what I needed - my system works.
(I still have no idea what is in the remainder of the 26 bytes!!
If I have missed some documentation then all advice is welcome!)

Below is Noise Reduction level request to the IC7000.


0xFE 0xFE 0x70 0xE0 0x1A 0x05 0x01 0x14 0xFD
0xFE 0xFE 0x70 0xE0 0x1A 0x05 0x01 0x14 0xFD

This is the TX request from piWeb to the IC7000
expanded to view the data decode.

The sequence is:

0xFE 0xFE - obligatory header bytes

0x70 - IC7000's address

0xE0 - address assigned to piWebCAT

0x05 0x01 0x14 the subcommand
(three bcd bytes for 050114)

0xFD - obligatory terminator

Web Browser Inspect element facility

This is a right mouse option on most web browsers.
It is particularly good on **Firefox developer** - and **Chrome**.

It provides a huge amount of information on traffic to and from websites.
It has very useful debugging facilities.

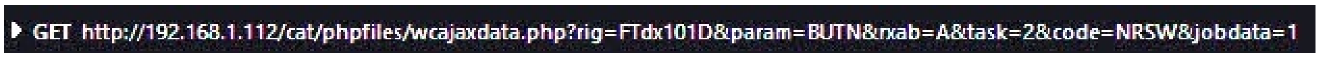
Its most useful application was in examining Ajax requests to the RPi server and the resulting responses.

If there an error in the server PHP code which is processing the request, the PHP echos back a detailed
error message containing the location of the problem

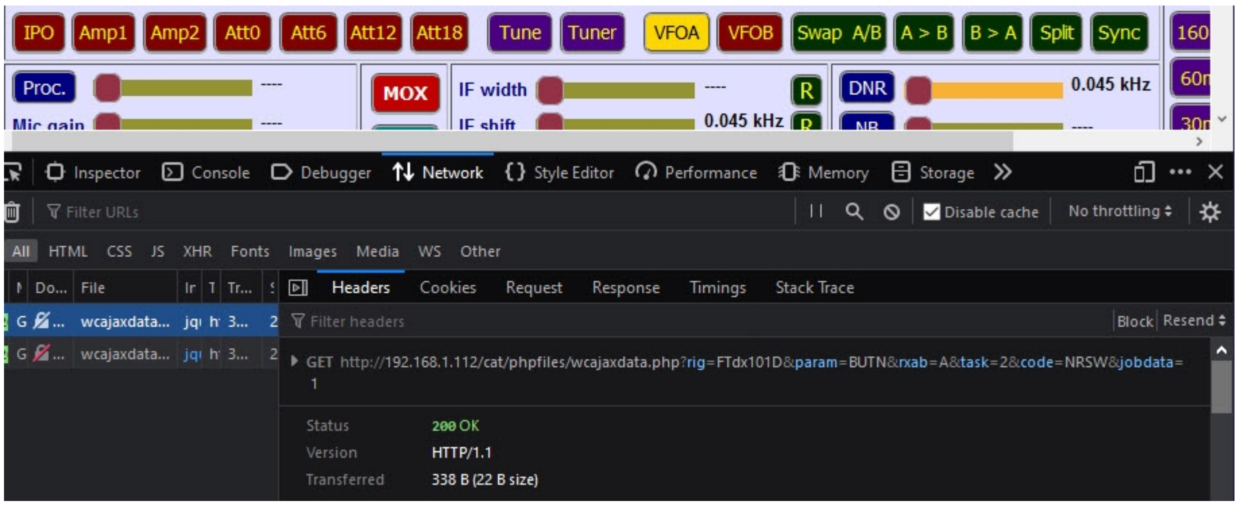
Firefox developer - Inspector - display examples

For the display below, I first disabled repetitive tasks in the timing table - network traffic comes to a stop.
- but the buttons are still active as evidenced by their effect on the radio.

I clicked right mouse over piWebCAT and then selected the network tab.
Then I clicked the DNR on/off button and the highlighted message appeared bottom left.
Then I selected the headers tab.
The display shows the complete URL sent to the server by Ajax code.
I have extracted the URL part and increased the contrast:

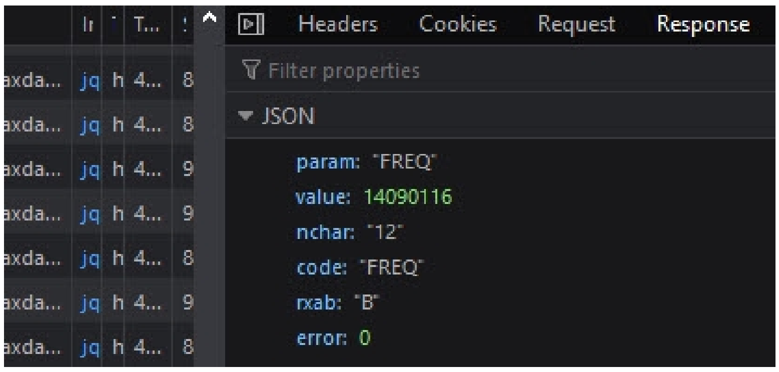


```
▸ GET  http://192.168.1.112/cat/phpfiles/wcajaxdata.php?rig=FTdx101D&param=BUTN&rxab=A&task=2&code=NRSW&jobdata=1
```



```
G  wcajaxdata...  jq  h  3...  2
G  wcajaxdata...  jq  h  3...  2

▸ GET http://192.168.1.112/cat/phpfiles/wcajaxdata.php?rig=FTdx101D&param=BUTN&rxab=A&task=2&code=NRSW&jobdata=
1

Status        200 OK
Version       HTTP/1.1
Transferred   338 B (22 B size)
```

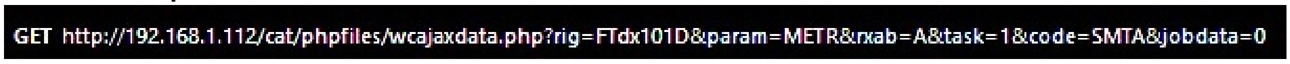
Further example - frequency read.



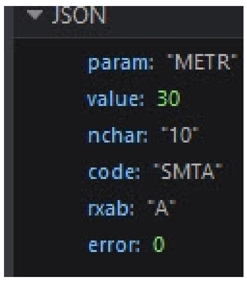
```
▾ JSON
    param: "FREQ"
    value: 14090116
    nchar: "12"
    code: "FREQ"
    rxab: "B"
    error: 0
```

For the display adjacent, I turned repetitive tasks back on -> A fast scroll of network traffic appears on the left.
piWebCAT is reading the S meter every 100mS and the frequency every 200ms.
I clicked on the fast scrolling data and froze one item - a frequency read.
The result is shown. Its a VFO B read of 14.090116 Hz

Further example - S meter read - FTdx101D



```
GET  http://192.168.1.112/cat/phpfiles/wcajaxdata.php?rig=FTdx101D&param=METR&rxab=A&task=1&code=SMTA&jobdata=0
```



```
▾ JSON
    param: "METR"
    value: 30
    nchar: "10"
    code: "SMTA"
    rxab: "A"
    error: 0
```

The above URL is a client to servers S meter level request.
Task=1 is code for read. jobdata = 0 because there is no data for a read request,

The response data is to the left.
The level is 30 (from a range of 0 -255)
The response contain's 10 ascii characters:
See answermask in meter table = RM0htu$$$;
htu is hundreds, tens and units of answer. $ are dont care characters

14.4 Configuring a RPi raspbian micro SD card.

A complete working micro SD card image is available for download.
The following pages describe how the card is set up.

We use ***terminal*** on the RPi.
nano is a text editor.   **sudo nano** gives the necessary root access

Initial - with monitor, mouse and keyboard

Download the latest Raspberry Pi operating system with desktop and software from:
https://www.raspberrypi.org/downloads/raspberry-pi-os/

(I do not use Noobs. It has unwanted operating systems in the image and more importantly, the image cannot be resized with ***parted.)***

Download and install Raspberry Pi Imager from:
https://www.raspberrypi.org/downloads/

Probably unnecessary... but:
Use windows - control panel - administrative tools -- computer management
- storage - disk management to delete all partitions from the card
(care! to not delete the wrong card /disk! ... I have done it!!)

Install raspbian OS on the card using Raspberry Pi Imager.

Insert the card in the RPi and power up

If your mouse is riduculously slow in response, the in RPi terminal do:

```
sudo nano /boot/cmdline.txt
```

Carefully add at the end (after a space) **`usbhid.mousepoll=0`**
Then save (Ctrl-X   Y   Enter)

Use start (bottom left)   Preferences - Raspberry Pi configuration - interfaces tab
Enable SSH, VNC, Serial port, remote GPIO

Configure keyboard and locale setting for you country.

Real VNC - will allow us PC control by Real VNC Viewer (free download for personal use)

The following may be needed to display the desktop a PC with Real VNC Viewer after you have disconnected the monitor from the RPi
(You may have already selected a display resolution - but you need to do it here!)

```
sudo raspi-config
```

Select the **advanced** option and then the **screen resolution** option.

If control by VNC is rejected then you will need the following change:

```
sudo nano /boot/config.txt
```

uncomment the line: **`hdmi_force_hotplug=`**1

(This will make the RPi generate HDMI output for VNC access
even if started with no HDMI monitor connected.)

Now check /etc/hosts

```
sudo nano /etc/hosts
```

make sure this line has changed to your host name

`127.0.1.1  piWebCAT` or whatever you chose

//

Now change to **fixed ip address**: I will use 192.168.1.117

Make sure dhcpcd is running:

```
sudo service dhcpcd start
sudo systemctl enable dhcpcd
```

```
sudo nano /etc/dhcpcd.conf
```

.... nano editor opens

.. assuming your router is 192.168.1.1
... 8.8.8.8 is a Google DNS .. for good measure !

Uncomment and modify this bit for wired networking.

```
interface eth0
static ip_address=192.168.1.117/24
static routers=192.168.1.1
static domain_name_servers=192.168.1.1   8.8.8.8
```

Add the following for WIFI - you can use the same IP address
The RPi will use wireless if eth0 is not connected

```
interface wlan0
static ip_address=192.168.1.117/24
static routers=192.168.1.1
static domain_name_servers=192.168.1.1   8.8.8.8
```

NB: you may choose to use a different IP address for wifi access.

////////////////

You may wish to continue with keyboard, mouse and monitor.
I disconnect them.
I install **Real VNC viewer** on the PC. (Free for private use but not for commercial use)
I display the RPi desktop on the PC
-- search 192.168.1.117 user pi password feline (save it)

Apache web server, php

```
sudo apt-get update

sudo apt-get install apache2 -y

sudo apt-get install php libapache2-mod-php -y
```

Check with web browser 192.168.1.117
Should show default apache2 demo index file

///

Pure-FTPd   used by Expression4 web developer   and by FileZilla FTP client

The following is mainly from Raspberry pi.org
Their example sets a home directory of / home/pi/FTP
We need a home directory of **/var/www/html** ... the apache webserver root

```
sudo apt-get install pure-ftpd

sudo groupadd ftpgroup
sudo useradd ftpuser -g ftpgroup -s /sbin/nologin -d /dev/null

sudo chown -R ftpuser:ftpgroup /var/www/html
```

user name is **upload** (or your choice)

```
sudo pure-pw useradd upload -u ftpuser -g ftpgroup -d /var/www/html -m
```

Enter and confirm the password.   I use: **feline**

```
sudo pure-pw mkdb

sudo ln -s /etc/pure-ftpd/conf/PureDB /etc/pure-ftpd/auth/60puredb

sudo service pure-ftpd restart
```

Now use FileZilla to check access:
  using: host = 192.168.1.117   user = upload   password = feline port = 21

.. this should access /var/www/html - the website root which at this
    point contains index.html (from the apache2 web server install)

Finally add another user to access the **/home/pi** folder

user name is **piuser** (or your choice)

```
sudo pure-pw useradd piuser -u ftpuser -g ftpgroup -d /home/pi -m
```

Enter and confirm the password.   I use: **feline**

```
sudo service pure-ftpd restart
```

You now have two users,

- **upload**   accesses   /var/www/html   - the website root
- **piuser**   accesses /home/pi   -   Downloads folder for Hamlib build etc

Finally - the upload process needs permission to access /home/pi and subfolders and files
```
sudo chmod 777 -R /home/pi
```

MariaDB (MYSQL) database

```
sudo apt-get install mariadb-server php-mysql -y
```

```
sudo apt-get install phpmyadmin
```

- **choose apache2** when asked which webserver reply **NO** to database question

```
sudo mysql -u root -p
```

- password will be requested **feline**

Your are now in MYSQL with a MYSQL prompt thus;

`MariaDB [(none)]>` you will now enter SQL statements ... ***all have terminating semicolon***.

```
CREATE DATABASE radios;

USE radios;

CREATE USER ‘piwebcat’@’localhost’ IDENTIFIED BY 'feline’;

GRANT ALL PRIVILEGES ON radios.* TO 'piwebcat'@'localhost' IDENTIFIED BY 'feline';

CREATE USER ‘piwebcat’@’%’ IDENTIFIED BY 'feline’;

GRANT ALL PRIVILEGES ON radios.* TO 'piwebcat'@'%' IDENTIFIED BY 'feline’;

FLUSH PRIVILEGES;
```

quit MYSQL

Edit /etc/mysql/my.cnf

```
sudo nano /etc/mysql/my.cnf
```

make sure bind-address=127.0.0.1 is commented out

add: **bind-address=0,0,0,0**
(To allow external access by MySQL Front etc)

finally:

```
 sudo service mysql restart
```

//

Edit /boot/cmdline.txt

```
sudo nano /boot/cmdline.txt
```

File content something like

```
console=serial0,115200 console=tty1 root=PARTUUID=75582189-02
 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash
 plymouh.ignore-serial-consoles
```

remove the section: **`console=serial0,115200`** and save

Edit boot/config.txt

```
sudo nano /boot/config.txt
```

comment out:

```
#dtparami2c_arm=on
#dtparam=spi=on
```

add under [all]

```
core_freq=250
enable_uart=1
dtoverlay=pi3-miniuart-bt
```

For reasons not understood, the serial ports looses access permissions
This is fixed by resetting them at start up
Edit .bashrc

```
    sudo nano /home/pi/.bashrc
```

Then go to end of this script and add:

```
    echo Running at end of script
    sudo chmod 666 /dev/ttyAMA0
    sudo chmod 666 /dev/ttyUSB0
    sudo chmod 666 /dev/ttyUSB1
```

The same can be achieved by creating file:

```
                     /lib/udev/rules.d/local.rules
```

Containing:

```
ACTION=="add", KERNEL=="dialout", MODE="0666"
ACTION=="add", KERNEL=="ttyAMA0", MODE="0666"
ACTION=="add", KERNEL=="ttyUSB0", MODE="0666"
ACTION=="add", KERNEL=="ttyUSB1", Mode="0666"
```

I do both !

```
sudo reboot
```

//

phpmyadmin

We have already installed **phpmyadmin**

To activate it and link to website root: /var/www/html

```
sudo ln -s /usr/share/phpmyadmin /var/www/html
```

```
sudo reboot
```

This installs a phpmyadmin file in the web site root

We can then access phpmyadmin from a browser by: **192.168.1.117/phpmyadmin**

user name is **piwebcat** password is **feline**

I have used MySQL Front for most of the development as it is PC based and perhaps quicker and easier to use that phpmyadmin. (PC only)
Recently I have used HeidiSQl which is an excellent alternative (PC based)

Download and install MYSQL Front

http://mysql-front.freedownloadscenter.com/windows/free/

Test connection using: host = 192.168.1.117 port = 3306 user piwebcat = pw = feline

- you should see the radios database
R mouse on it click - import - select my supplied radios.sql - Run
- whole database should quickly import

Use FileZilla to upload the website to /var/www/html

(The FTP server, pure-ftpd is already configured with /var/www/html as upload root.

host = 192.168.1.117 port = 21 user = upload password = feline

14.5 Installing /updating Hamlib on the SD card

Using the web browser ***on the RPi***, access: https://github.com/Hamlib/Hamlib

CODE button - download zip (zipped source code)
- down load to /home/pi/Downloads (or other folder of your choice)

File manager - navigate to folder /home/pi/downloads
click on Hamlib-master.zip and unzip to same folder.

Terminal on RPi - navigate to Hamlib-master folder, ie:

```
$    cd Downloads
$    cd Hamlib-master
```

*You need to have made the following download/installations **once**.*
You will not need to run these installs on subsequent Hamlib builds

```
$    sudo apt-get install automake
$    sudo apt-get install libtool
```

- bootstrap permissions;

`$   chmod 744 bootstrap` (this is not remembered unless you configure it in startup)

```
$    ./bootstrap

$   ./configure --prefix=/usr/local --enable-static

$   make

$   sudo make install

$   sudo ldconfig
```

Test the install. This will give a version report.

```
$   rigctl -V
```

If you now run **File manager** from the RPi desktop icon, you display the contents of **/home/pi**.

You have FTP access to /home/pi from an FTP client (eg: Filezilla) on you PC using:
IP address (eg: 192.168.1.117, user = piuser, password = feline.
Enter the Downloads directory. You will see Hamlib-master.zip and the Hamlib-master folder.
These are no longer needed once the above install is completed.
They should be deleted before any further download / installs of Hamlib.

Give the apache webserver the rights to run killall frrom a shell_exec command
RPi terminal - edit the sudoers file:

```
$     sudo nano /etc/sudoers
```

Add the line: `apache localhost=(ALL) NOPASSWD:/usr/bin/killall`

This allows the php code to execute: `shell_exec("killall rigctld &");`

Explanation: If you change from one Hamlib rig to another. then a restart occurs.
On the server, **rigctld** will not restart if it is still running from the previous session.
The system will therefore freeze when it attempts to get responses from the previous (wrong) rig.
piWebCAT PHP calls: `shell_exec("killall rigctld &");` to kill rigctld before restarting it.
It needs the above permission change in the /etc/sudoers file to allow it to do this.

14.6 Adding Mumble server and client to the micro SD card

Mumble server

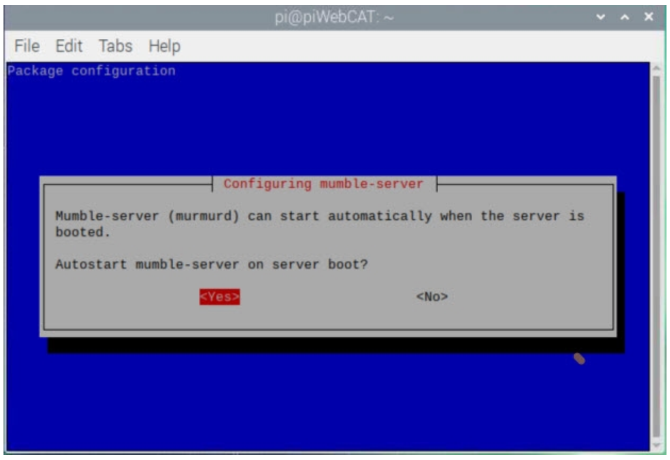
RPi terminal:

```
sudo apt-get install mumble-server --fix-missing
```

`sudo dpkg-reconfigure mumble-server` (This can be re-run any time)

A succession of windows appear (navigate with arrow keys then Enter - mouse doesn't work)

The are three question windows. First one shown


```
Package configuration

Configuring mumble-server

Mumble-server (murmurd) can start automatically when the server is
booted.

Autostart mumble-server on server boot?

<Yes>                    <No>
```

Autostart on boot - **Yes**

Higher priority - **Yes**

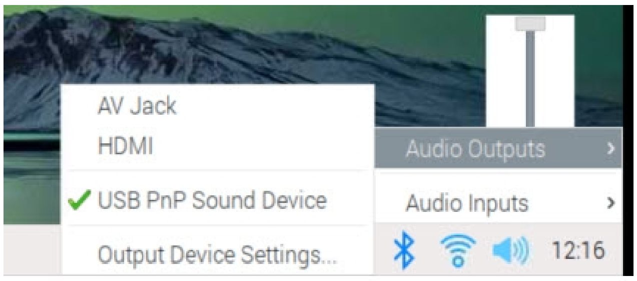
Password - your choice - I set **feline** on SD card

```
sudo /etc/init.d/mumble-server restart
```

Mumble client on RPi

```
sudo apt-get install mumble --fix-missing
```

At this point - make sure that the USB audio adapter is plugged in to the RPi.



Click the speaker icon taskbar-right - set max volume.

Right click the speaker icon:
- Select USB PnP Sound Device as output
- Select USB PnP Sound Device as input
 (? the only option)

Click the raspberry icon task bar left. Scroll up to Internet, across to Mumble - right mouse - add to desktop.

Mumble can now be started with the newly added Mumble desktop icon (or in RPi terminal type: **`mumble)`**

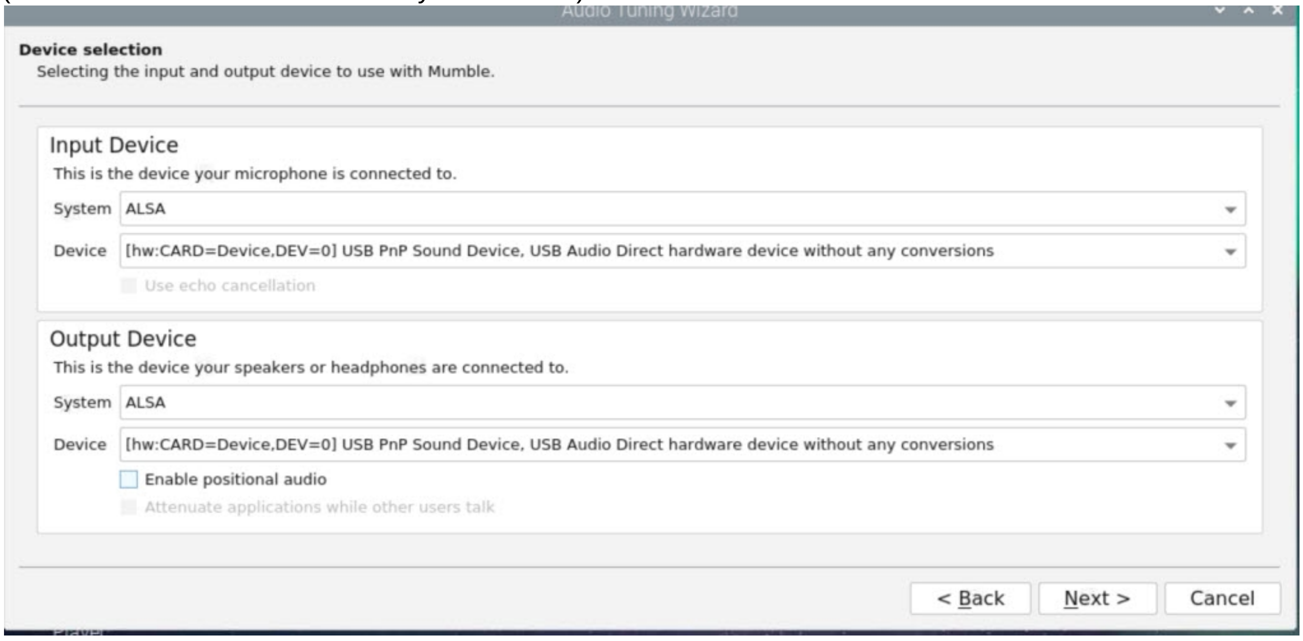
Mumble should launch screen top left with a Welcome screen.
(if not Welcome screen then use configure - audio wizard)

Mumble client on RPi continued.....

- click **Next**

You now make some configuration selections. *These can be adjusted later by simply re-running mumble.*

An input / output device selector window appears - configure as shown below:
(Make sure to select ..... without any conversions)

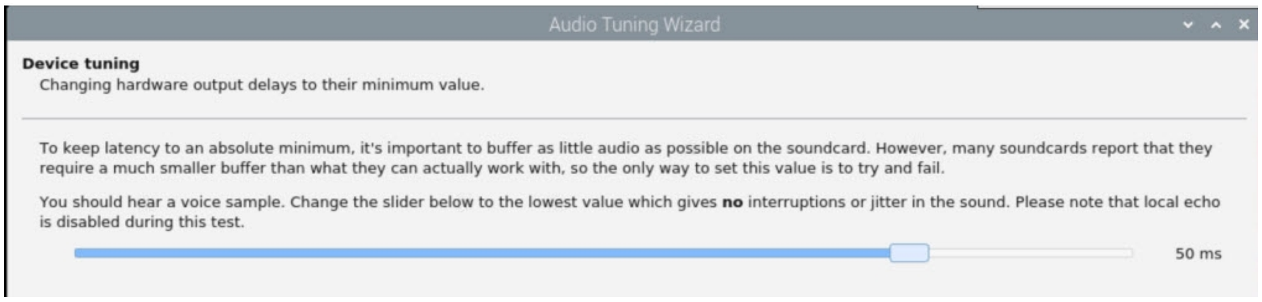


Device selection
Selecting the input and output device to use with Mumble.

Input Device
This is the device your microphone is connected to.
System ALSA
Device [hw:CARD=Device,DEV=0] USB PnP Sound Device, USB Audio Direct hardware device without any conversions
Use echo cancellation

Output Device
This is the device your speakers or headphones are connected to.
System ALSA
Device [hw:CARD=Device,DEV=0] USB PnP Sound Device, USB Audio Direct hardware device without any conversions
Enable positional audio
Attenuate applications while other users talk

- Click **Next**
Plug headphones into adapter output.
The **Device tuning** latency window appears - follow instructions.
(If no voice output, leave at 50ms for now)



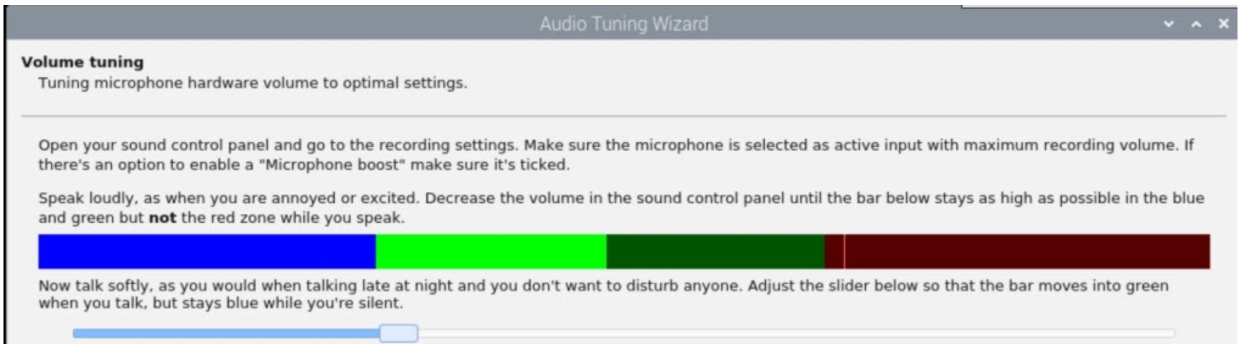
Device tuning
Changing hardware output delays to their minimum value.

To keep latency to an absolute minimum, it's important to buffer as little audio as possible on the soundcard. However, many soundcards report that they require a much smaller buffer than what they can actually work with, so the only way to set this value is to try and fail.

You should hear a voice sample. Change the slider below to the lowest value which gives **no** interruptions or jitter in the sound. Please note that local echo is disabled during this test.

50 ms

Mumble client on RPi continued.....
Click **Next**:
The **Volume tuning** window appears as below.
This is a volume setting for transceiver Rx output to the USB sound adapter 'microphone' input.
.. So it has nothing to do with actual microphones!
You need to connect the adapter input to the transceiver Rx AF output.
If this is from a rear panel data connector, then it is likely to be at an AGC determined fixed level for a medium/strong received signal level. (eg: The FTdx101D has a 300mV output, not controlled by AF gain.)


Audio Tuning Wizard

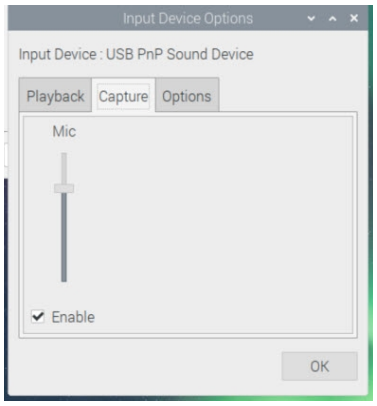
Volume tuning
Tuning microphone hardware volume to optimal settings.

Open your sound control panel and go to the recording settings. Make sure the microphone is selected as active input with maximum recording volume. If there's an option to enable a "Microphone boost" make sure it's ticked.

Speak loudly, as when you are annoyed or excited. Decrease the volume in the sound control panel until the bar below stays as high as possible in the blue and green but **not** the red zone while you speak.

Now talk softly, as you would when talking late at night and you don't want to disturb anyone. Adjust the slider below so that the bar moves into green when you talk, but stays blue while you're silent.

Right click the speaker icon again and then select **Audio input and** then **Input device options** as shown below..
Select the **Capture** tab and make sure **Mic input** is enabled.

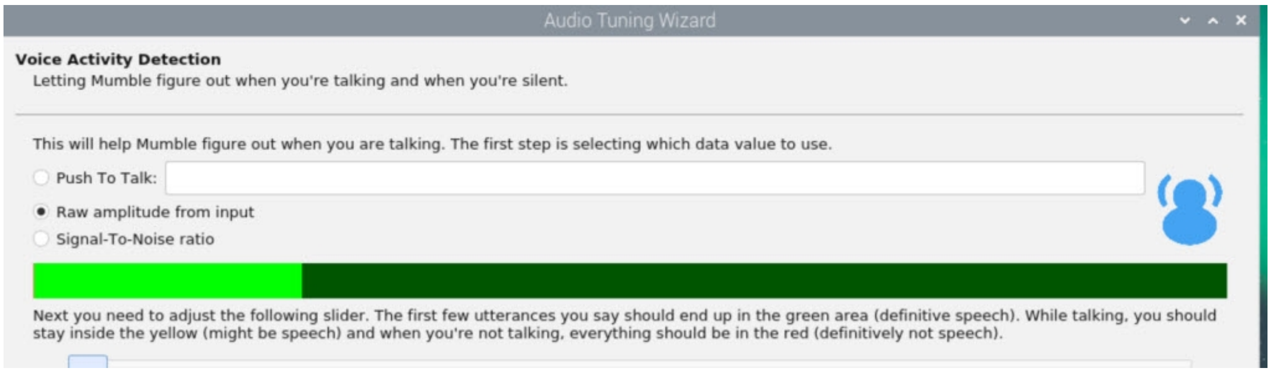


The supplied SD card RPi installation doesn't provide this window frrom clicking the speaker icon.
Earlier installations did show it.

I made a complete SD card rebuild for final issue to users in order to have the latest RPi operating system and to avoid any clutter from nine months of development.
Itdoesn't seems to matter - Mumble worked fine

Feed in received audio from the transceiver and adjust the slider in Input Device Options so that the thin marker bar
is at the high end of the green section. (This corresponds to the displayed 'speak loudly' instruction.)
Then reduce the volume to a low level (probably by reducing RF gain) and adjust the slider on the mumble window
for the marker bar to stay in the blue (This corresponds to the 'speak softly' instruction.)

Click **Next**
The **Voice Activity Detection** window appears.
This appears to be a Mumble VOX system. We don't need it. This is received audio!
Select **Raw amplitude from input** and push the slider fully left so that mumble thinks that a signal is always present.


Audio Tuning Wizard

Voice Activity Detection
Letting Mumble figure out when you're talking and when you're silent.

This will help Mumble figure out when you are talking. The first step is selecting which data value to use.

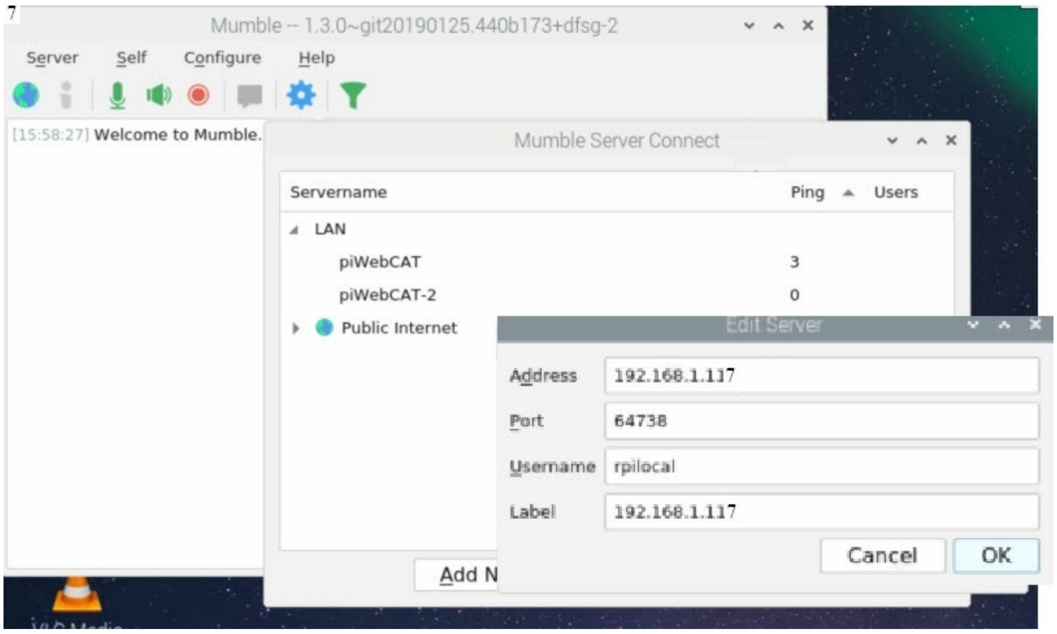
○ Push To Talk:
● Raw amplitude from input
○ Signal-To-Noise ratio

Next you need to adjust the following slider. The first few utterances you say should end up in the green area (definitive speech). While talking, you should stay inside the yellow (might be speech) and when you're not talking, everything should be in the red (definitively not speech).

Mumble client on RPi continued.....
Click **Next**:
The **Quality & Notifications** window appears.
Select **High quality** and **Disable Text-to-Speech**

Click **Next**:
A **Finished** window appears
... You have the option of whether to submit anonymous statistics to the Mumble project.
I disabled this because our reversal of Rx and Tx might produce confused statistics?
Click **Finish**

Now we have to **set up a connection to the mumble-server on this RPi.**
With Mumble still running - this is client configuration. We have to connect it to the mumble server on this RPi.
Click menu - **Server** and then **Connect.**
Then click **Add New...**



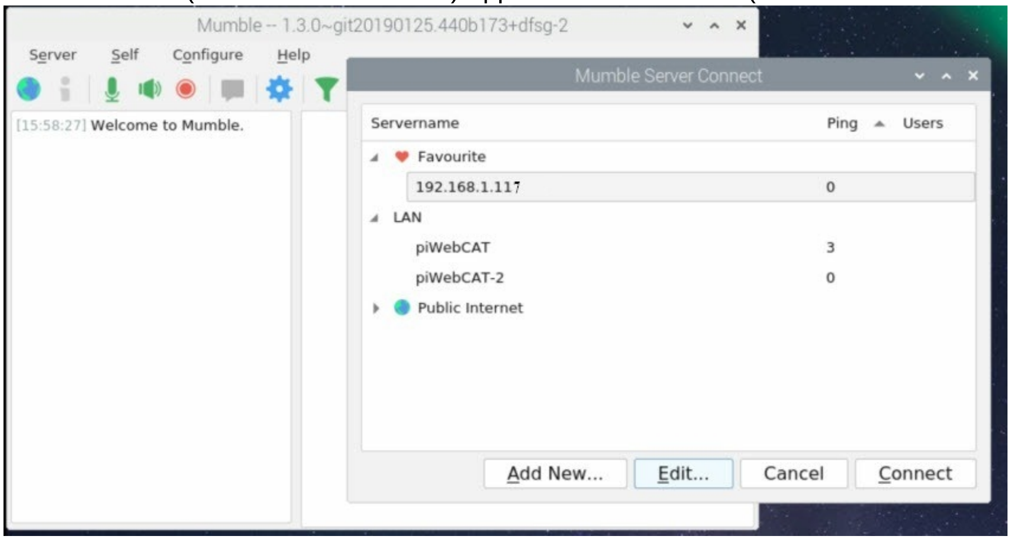
[15:58:27] Welcome to Mumble.

Servername	Ping	Users
piWebCAT	3	
piWebCAT-2	0	

Address: 192.168.1.117
Port: 64738
Username: rpilocal
Label: 192.168.1.117

Enter in the Address box the ***IP address of this RPI***, eg: 192.168.1.117 above
Port **64738** should appear by default.
Give the connection a name - eg: I suggest **rpilocal** here (The supplied micro SD card has **rpilocal**)
The name of the connection defaults to the IP address (you can change it)

Click **Ok**:
The connection (named 192.168.1.117) appears as a favourite. (The connection can be edited later.)

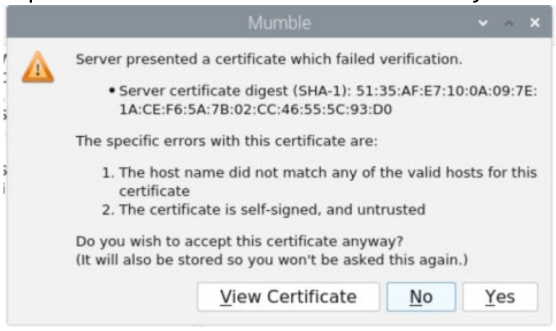

Mumble -- 1.3.0~git20190125.440b173+dfsg-2
Server Self Configure Help
Mumble Server Connect
Servername Ping Users
Favourite
LAN
Public Internet
Add New... Edit... Cancel Connect
[15:58:27] Welcome to Mumble.

Servername	Ping	Users
192.168.1.117	0	
piWebCAT	3	
piWebCAT-2	0	

Mumble client on RPi continued.....

Click **Connect**:

You are presented with a window that informs you that certificate verification has failed.



Server presented a certificate which failed verification.

- Server certificate digest (SHA-1): 51:35:AF:E7:10:0A:09:7E:1A:CE:F6:5A:7B:02:CC:46:55:5C:93:D0

The specific errors with this certificate are:

1. The host name did not match any of the valid hosts for this certificate
2. The certificate is self-signed, and untrusted

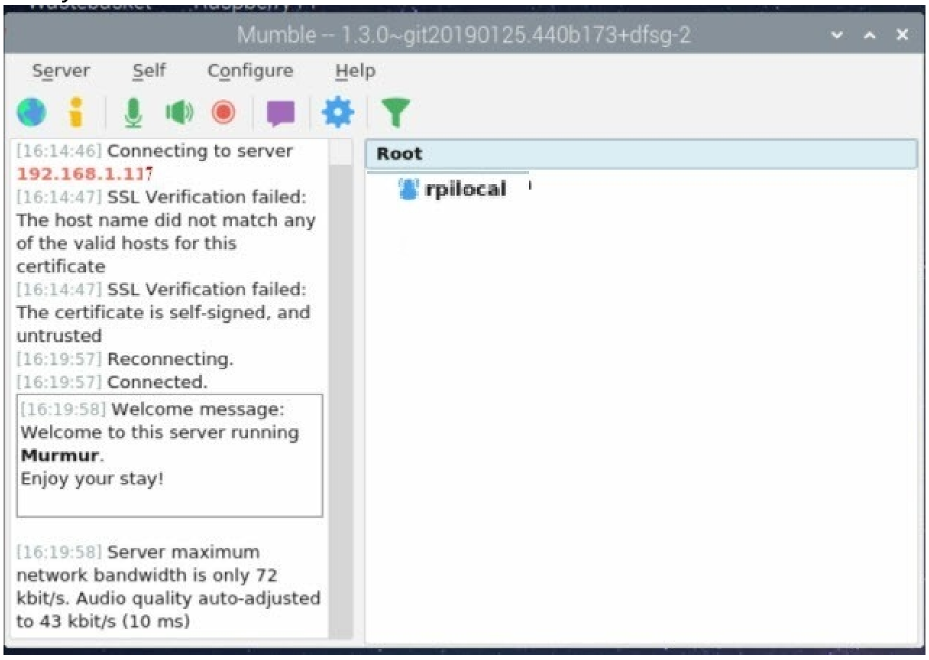
Do you wish to accept this certificate anyway?
(It will also be stored so you won't be asked this again.)

Click **Yes** to accept the certificate.

You will not then be presented with this failure message when you connect again.

Note that password login is an option - but we don't want this.
We want the complete RPi server and client system to auto start on power up without any user interaction.

Finally - we are connected



```
[16:14:46] Connecting to server 192.168.1.117
[16:14:47] SSL Verification failed: The host name did not match any of the valid hosts for this certificate
[16:14:47] SSL Verification failed: The certificate is self-signed, and untrusted
[16:19:57] Reconnecting.
[16:19:57] Connected.
[16:19:58] Welcome message: Welcome to this server running Murmur.
Enjoy your stay!
[16:19:58] Server maximum network bandwidth is only 72 kbit/s. Audio quality auto-adjusted to 43 kbit/s (10 ms)
```

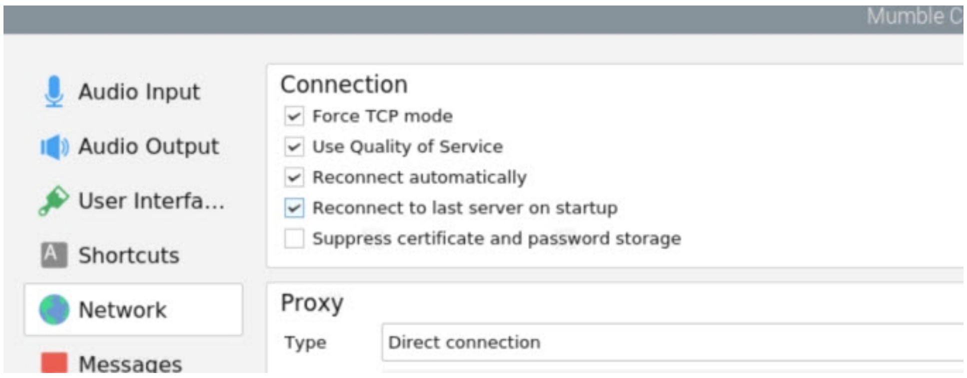
Mumble client on RPi continued.....

We now need to ensure that the **whole system starts automatically on RPi power up**.

The **mumble-server** is dealt with - It was configured for auto-start on reboot.

For the client:
Click the **Configure** menu option again and then click the **Network** button.

Reconnect automatically should already be checked - see below.
Check the **Reconnect to last server on Startup** box.
Check the **Force TCP mode** box.
Ensure that **Direct connection** is selected in the Proxy panel (No proxy server)



Connection
- [✓] Force TCP mode
- [✓] Use Quality of Service
- [✓] Reconnect automatically
- [✓] Reconnect to last server on startup
- [ ] Suppress certificate and password storage

Proxy
Type: Direct connection

Click **Apply** and then **OK.**

Now close Mumble using menu **Server** ... **Quit** and then restart it
(Otherwise your configuration will not be saved)

We are now in a position where clicking the mumble icon on the desktop will start mumble and activate all the audio connections.
Finally we must configure auto start of Mumble on power up / reboot.

Using RPi terminal type:

```
sudo nano /etc/xdg/autostart/mumblestart.desktop
```

This opens the nano editor to create a new file. The file is therefore initially empty
Enter the following text into the file:

```
[Desktop Entry]
Type=Application
Name=Mumble
Comment=Audio VOIP client
NoDisplay=true
Exec=mumble
NotShowIn=GNOME;KDE;XFCE;
```

Then type the usual **Ctrl-X Y** and then **Enter** to save the file.

To test everything:

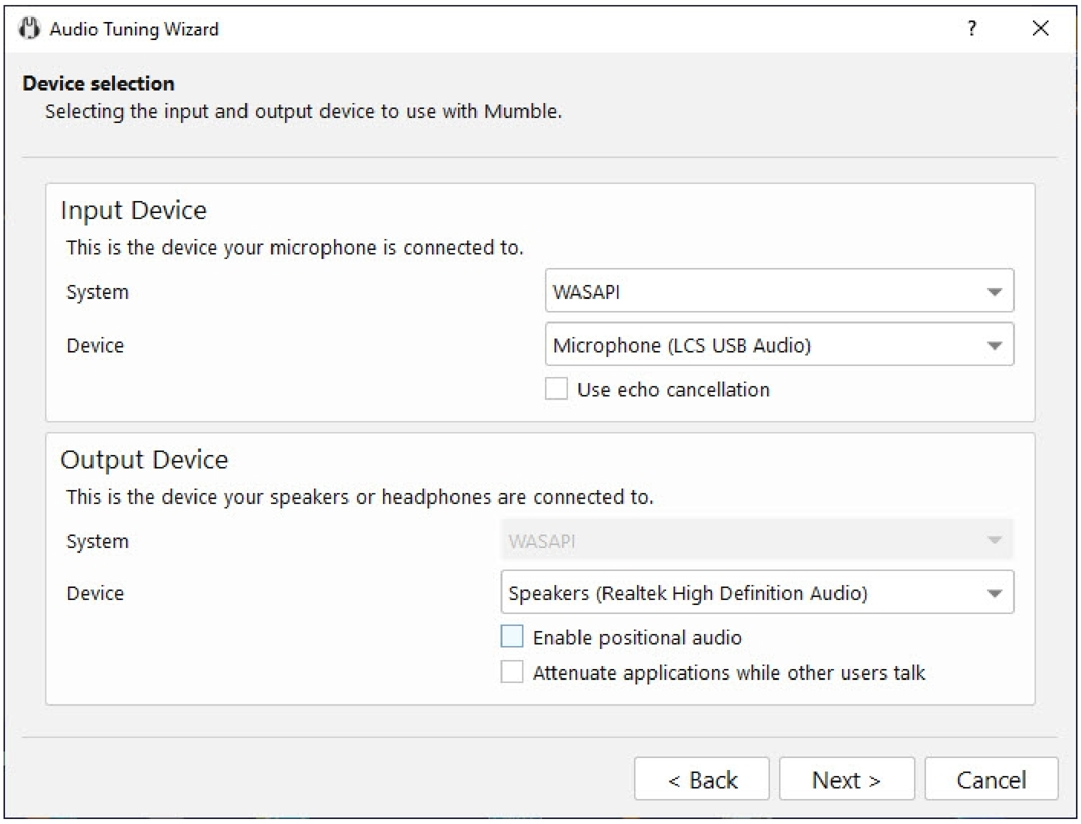
```
sudo reboot
```

The RPi should restart with mumble visible and operational.

14.7 Installing Mumble on a controlling device - Windows PC

Navigate to https://www.mumble.info. Click the **Download now** button and the download the installer.
Run the Mumble installer (currently mumble-1.3.3.winx64.msi for a 64 bit machine.)
The default is to only install the client. Do not add in server installation - The server is on the RPi.
If you will be using an external microphone, then plug it in.

Start Mumble .. a welcome window appears. Click **Next**.
A Device selection window appears. This is the same as for the RPi above - but with different devices,
The window below was on my 17inch Levono laptop.
I am using here an external USB microphone and the internal speakers.


Audio Tuning Wizard

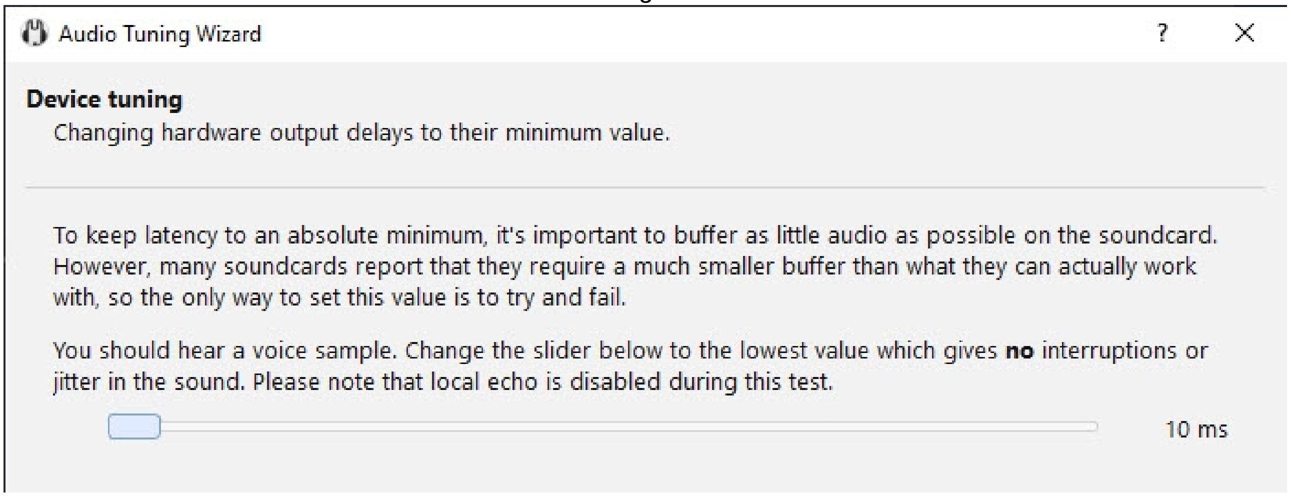
Device selection
Selecting the input and output device to use with Mumble.

Input Device
This is the device your microphone is connected to.
System: WASAPI
Device: Microphone (LCS USB Audio)
☐ Use echo cancellation

Output Device
This is the device your speakers or headphones are connected to.
System: WASAPI
Device: Speakers (Realtek High Definition Audio)
☐ Enable positional audio
☐ Attenuate applications while other users talk

Click **Next**

Follow the instructions . I was able to reduce the setting to the minimum of 10ms.


Audio Tuning Wizard

Device tuning
Changing hardware output delays to their minimum value.

To keep latency to an absolute minimum, it's important to buffer as little audio as possible on the soundcard. However, many soundcards report that they require a much smaller buffer than what they can actually work with, so the only way to set this value is to try and fail.

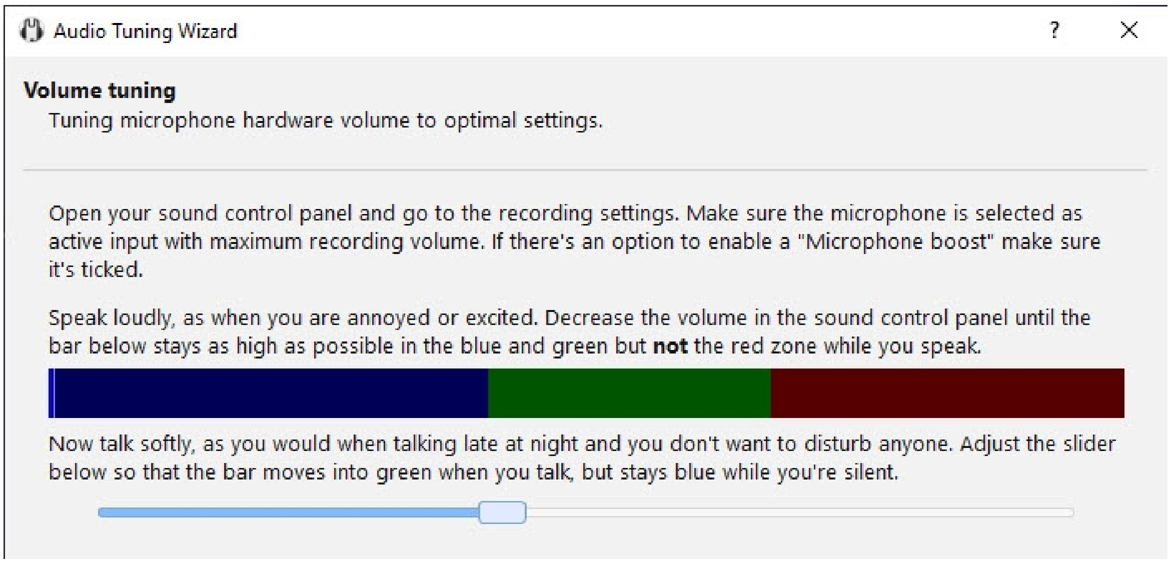
You should hear a voice sample. Change the slider below to the lowest value which gives **no** interruptions or jitter in the sound. Please note that local echo is disabled during this test.

10 ms

Mumble on controlling device Windows PC - continued

Click **Next**

The Microphone Volume Tuning window appears *(This time it IS the microphone)*
Follow the instructions. Don't worry if you can't achieve enough volume here.


Audio Tuning Wizard

Volume tuning
Tuning microphone hardware volume to optimal settings.

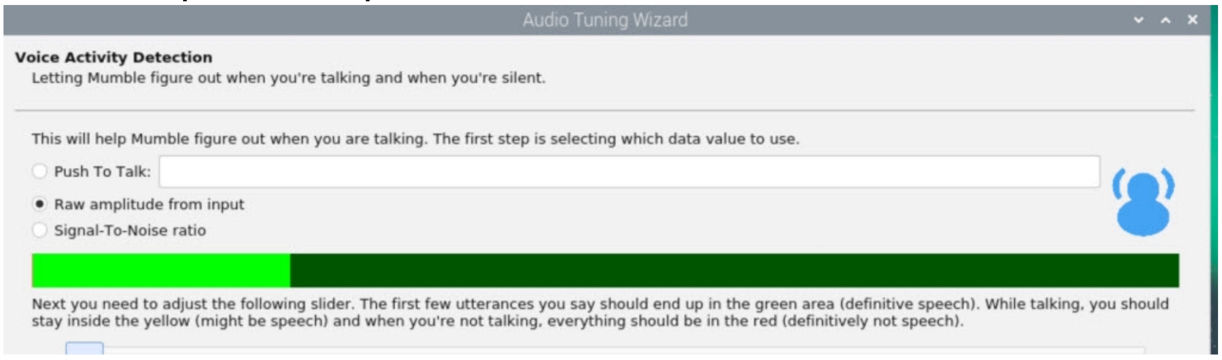
Open your sound control panel and go to the recording settings. Make sure the microphone is selected as active input with maximum recording volume. If there's an option to enable a "Microphone boost" make sure it's ticked.

Speak loudly, as when you are annoyed or excited. Decrease the volume in the sound control panel until the bar below stays as high as possible in the blue and green but **not** the red zone while you speak.

Now talk softly, as you would when talking late at night and you don't want to disturb anyone. Adjust the slider below so that the bar moves into green when you talk, but stays blue while you're silent.

Click **Next**

Voice Activity Detection - This is mumble VOX - we don't need it.
Push slider fully to left.
Select **Raw amplitude from input**.


Audio Tuning Wizard

Voice Activity Detection
Letting Mumble figure out when you're talking and when you're silent.

This will help Mumble figure out when you are talking. The first step is selecting which data value to use.

Push To Talk:
[✓ Raw amplitude from input]
Signal-To-Noise ratio

Next you need to adjust the following slider. The first few utterances you say should end up in the green area (definitive speech). While talking, you should stay inside the yellow (might be speech) and when you're not talking, everything should be in the red (definitively not speech).

Click **Next**
The **Quality & Notifications** window appears

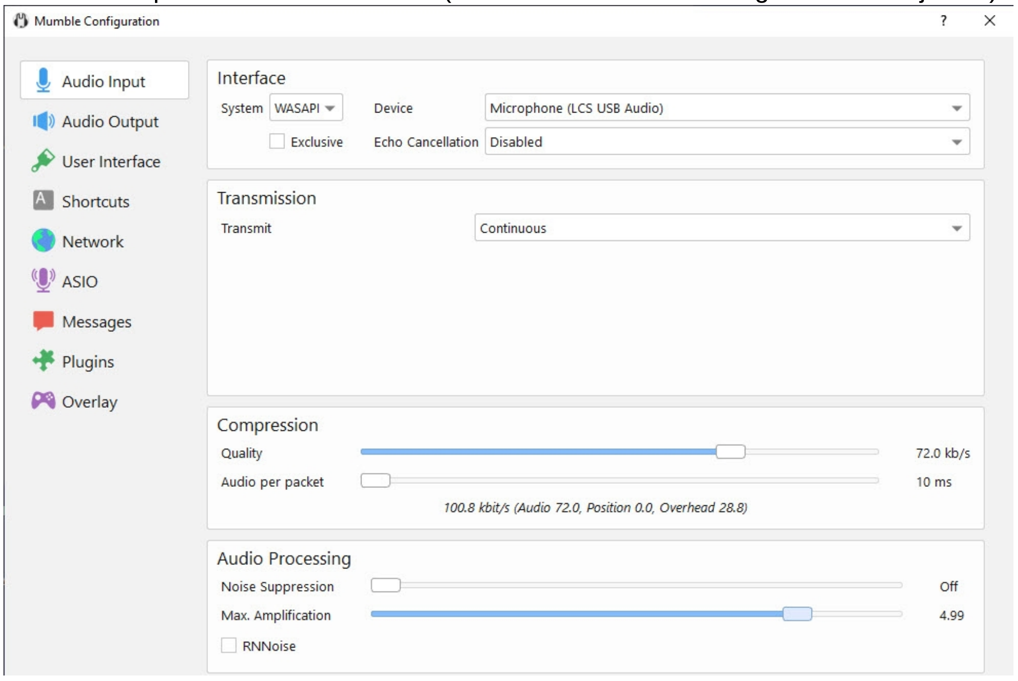
Select **High quality** and **Disable Text-to-Speech**

Click **Next**:

A **Finished** window appears
You have the option of whether to submit anonymous statistics to the Mumble project.

Click **Finish**

Mumble on controlling device Windows PC - continued
On the Mumble menu Click **Configure - Settings.**
Select the **Audio input** tab.   Select Transmit = **continuous**
Set compression quality = 72 kb/s.   Set Noise suppression = Off.
Set Max. Amplification = 5.0 for now. (You can return here if the gain has to adjusted)



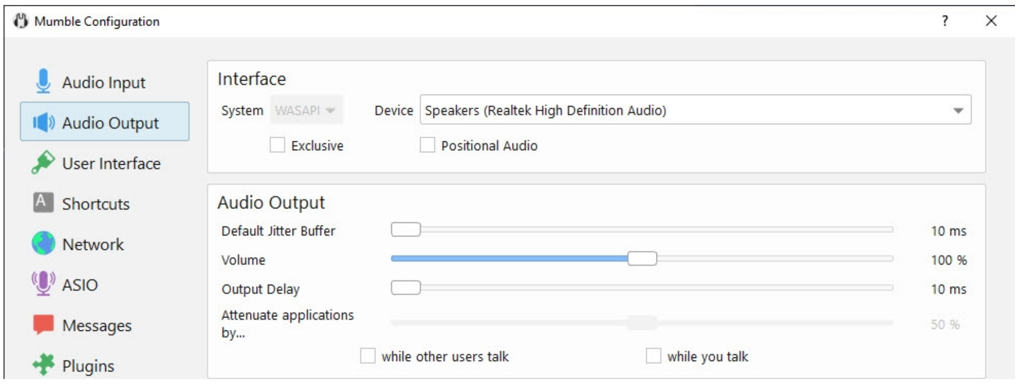
Interface
System WASAPI   Device Microphone (LCS USB Audio)
Exclusive   Echo Cancellation Disabled

Transmission
Transmit Continuous

Compression
Quality 72.0 kb/s
Audio per packet 10 ms
100.8 kbit/s (Audio 72.0, Position 0.0, Overhead 28.8)

Audio Processing
Noise Suppression Off
Max. Amplification 4.99
RNNoise

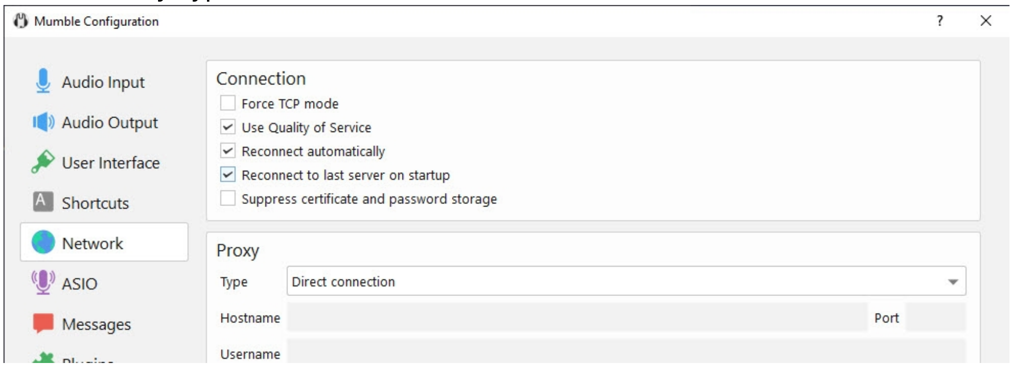
Select the **Audio Output** tab
Leave volume at 100%



Interface
System WASAPI   Device Speakers (Realtek High Definition Audio)
Exclusive   Positional Audio

Audio Output
Default Jitter Buffer 10 ms
Volume 100 %
Output Delay 10 ms
Attenuate applications by... 50 %
while other users talk   while you talk

Select the **Network** tab
Check **Reconnect last server on startup**.
Ensure 'Proxy' type is **Direct connection**



Connection
- [ ] Force TCP mode
- [x] Use Quality of Service
- [x] Reconnect automatically
- [x] Reconnect to last server on startup
- [ ] Suppress certificate and password storage

Proxy
Type Direct connection
Hostname   Port
Username

There are other configuration settings that you may wish to alter later (eg: appearance of mumble user interface)
Click Accept and then **Ok.**
Close Mumble with menu **Server - Quit Mumble** and then restart it. (*This is necessary to save the settings*)

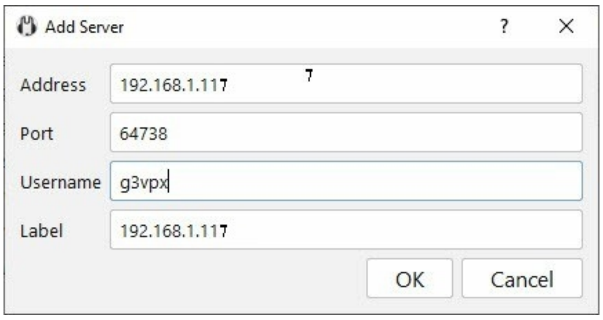
Mumble on controlling device Windows PC - continued

Connecting to the server on the RPi.

Click menu - **Sever** and then **Connect.**
Then click **Add New...**

Insert the RPi address of the RPi.
(Also appears as the label
- which you can change)
The default port should be left at
the Mumble default of 64738.

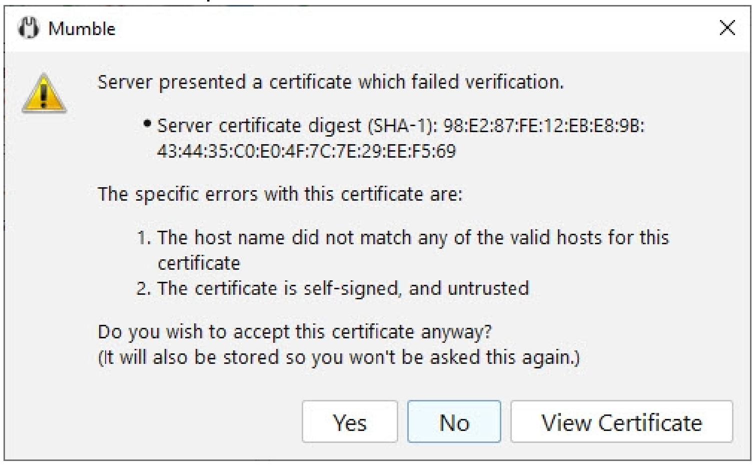
Enter a Username of your choice,
eg: your callsign.
Click OK.



Add Server

Address: 192.168.1.117
Port: 64738
Username: g3vpx
Label: 192.168.1.117

The server is added as a **Favourite.**
Select this server and click **Connect.**
On first connect you are presented with a failed certificate message.
Click **Yes** to accept it



Mumble

Server presented a certificate which failed verification.

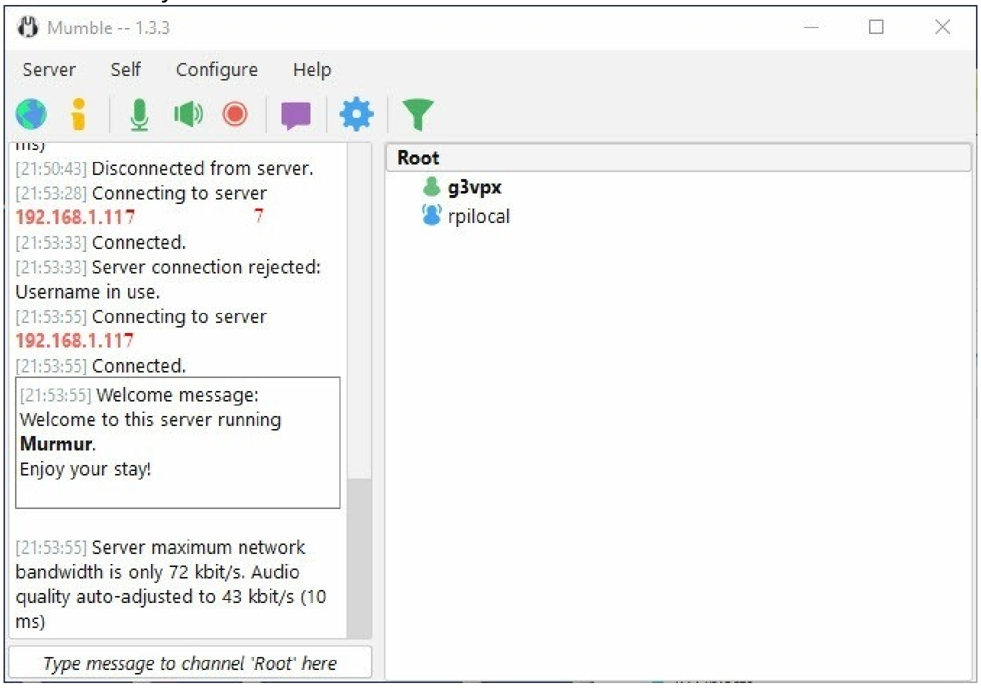
- Server certificate digest (SHA-1): 98:E2:87:FE:12:EB:E8:9B:43:44:35:C0:E0:4F:7C:7E:29:EE:F5:69

The specific errors with this certificate are:

1. The host name did not match any of the valid hosts for this certificate
2. The certificate is self-signed, and untrusted

Do you wish to accept this certificate anyway?
(It will also be stored so you won't be asked this again.)

Your are finally connected!



Mumble -- 1.3.3

```
ms)
[21:50:43] Disconnected from server.
[21:53:28] Connecting to server
192.168.1.117
[21:53:33] Connected.
[21:53:33] Server connection rejected:
Username in use.
[21:53:55] Connecting to server
192.168.1.117
[21:53:55] Connected.
[21:53:55] Welcome message:
Welcome to this server running
Murmur.
Enjoy your stay!

[21:53:55] Server maximum network
bandwidth is only 72 kbit/s. Audio
quality auto-adjusted to 43 kbit/s (10
ms)
```

Root
- g3vpx
- rpilocal

14.8 Installing Mumble on a controlling device - Android / IOS

re: Apple IOS; I have not experimented with Mumble for Apple IOS.
I no longer have any Apple devices. I donated my ageing Ipad Mini to a local school IT department following
the request for tablets and laptops for use in the covid lockdown!

A search of Google play Store for 'mumble' reveals Mumble client apps: Plumble and Mumla in various offerings.

I installed Plumble. The appear to be two versions entitled: Plumble - free and Plumble - Mumble VOIP

The **Plumble - Mumble VOIP** version appears to be free - I installed this on a modern Vankyo 10 inch tablet.

Before installing, make sure that the RPi is running with the mumble-server active.

Start the app.
As soon as the app starts you are offered the creation of a certificate - tap Yes to this.

Then tap the **+** icon top right to add a server.
Enter the RPi server IP address, eg: 192.168.1.117.
Accept the default port = 64738.
Add a label and username of you choice.
Leave the password field blank.

A connection box will appear top left - Tap the connection box to connect.

An **Untrusted certificate** box appears - tap **Allow** to accept it.

With the transceiver running I had immediate Rx audio.

Now click the menu ion top left and then **Settings**

General tab: Uncheck: Chat notifications, text to speech and load external images
Check: Auto Reconnect

Audio tab: Transmit mode - set this to **continuous** (This will disable many of the voice control options)
Handset mode: - uncheck this
Microphone volume: You need to adjust this for your rig's ALC / comp indications
(mine is set to 30% ... using the tablets built in microphone)

Detection threshold, PTT key, PTT Hot Corner, PTT Sound,
Hide PTT button, Toggle PTT, Half duplex mode
- all these are disabled because we set Transmit mode = continuous.

Input sample rate: - The 11025Hz option should be adequate for voice
Input quality: I set 72000bps to match what was recommended for RPi client setup.
Enable speech processor: No ? - We can experiment with this.
Disable Opus Codec: No

14.9 Javascript debug popup window

This window displays nine data items, each with an index (1 to 9) and a text label.

It is launched by any spare button control of your choice configured with:

- **code = DBUG** This is a fixed code linked to the debug popup.
- **active = Y**
- **action = S** A single shot button - as used to launch the MPAD and MORE popups.

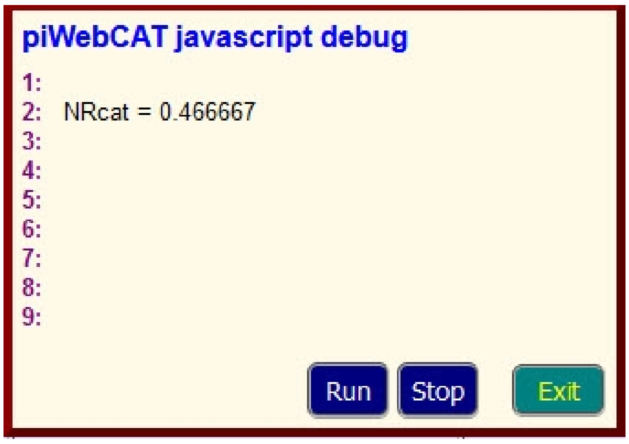
Debug items are displayed from running code by temporarily inserting code of the form:

```
pwcDebug(3, 'mylabel',  data);
```

This will update item 3 on the popup with: **mylabel = data**.

The example below results from code inserted in slider read callback in javascript module sliders.js .

```
if(code=='NRLV'){pwcDebug(2,'NRcat',cat);}
```


piWebCAT javascript debug

1:
2: NRcat = 0.466667
3:
4:
5:
6:
7:
8:
9: