Qt-DAB 6.*

User's guide for version 6.5

Jan van Katwijk, Lazy Chair Computing
The Netherlands

January 31, 2024

Contents

1 Qt-DAB-6.5

1.1 Introduction

Qt-DAB is an advanced program for decoding terrestrial DAB and DAB+ transmissions. T program decoded a DAB stream, input is by an *SDR* device or from a file. Qt-DAB is (or can be) configured with a number of popular SDR devices, such as SDRplay RSP's, AIRspy, Lime, Hackrf, Adalm pluto, and obviously, the cheap RTLSDR devices (including the recent V4 version), the dabsticks. Furthermore Qt-DAB can get its input using rtl_tcp or from a *spyserver*.

Qt-DAB is designed to run on Linux (x64) computers, on RPI 3 and up running some Linux variant, and is *cross compiled* for Windows (32 bits).

The focus in the development always is - next to actual decoding the signal of course - visualisation of the signal in its raw and its processed form. This is reflected in the structure of the GUI.

The GUI of Qt-DAB consists of 4 parts:

- the *main* widget, the main widget contains - next to ontrol to select channels and services - the controls for visibility of the other three widgets;
- the *configuration and control* widget, contains a large amount of controls for different settings in the program;
- a *spectrum scope widget*, containing a tabwidget with the choice of one of 6 difference views on the signal;
- *technical details widget*, showing the technical details of the audio of the selected service.

Qt-DAB-6.5 succeeds previous versions with a number of improvements, apart from numerous smaller changes internally, two of which are (a) an integration of the favorites in the ensemble list, and (b) the support of spyserver input.

The structure of this guide is simple, section 2 deals with installing the program, in section 3 the GUI and GUI widgets are described, In section 4 some functions, all related to scanning and showing the map, are discussed. in section 5 *Scheduling* of service selections is discussed, and, finally, in section 6 the supported devces for the Qt-DAB program are briefly discussed.

1.2 Hardware Requirements

Qt-DAB runs pretty well on modern PC's. One does need a reasonable amount of computing power, though. On an RPI 3 and up the program will run well, obviously on an Arduino it will not. The load on Windows is app 5 to 10 times higher than the load on Linux x64, on a 5 year old laptop running W10 (With a 2.5G I5 processor) the load

is app 30 percent, on my newer x64 laptop running Linux, also running an I5 (10-th generation), takes between 2 and 3 percent.

2 A note on installing Qt-DAB-6.5

Two windows installers for Qt-DAB-6.5 is available. Each installer will install a version of Qt-DAB by creating a folder in the system folder "Program Files (x86)" on the C drive and will create - if checked - a link to the desk.

The difference between the versions is the support for RT2832 based sticks, while the V3 version is equipped with the classic (osmocom) support software for these sticks, the other version is equipped with support software for the V4 version of the stick. It seems that using an older stick with the V4 version library the software was "deaf".

For Linux on the x64 PC, a so-called *AppImage* is available. Such an AppImage is itself a small closed filesystem with all programs and libraries for running Qt-DAB aboard. However, the AppImage does NOT contain support libraries for the various configured devices. On selecting a device, Qt-DAB will look for a device support library, and will load the relevant functions if found. If not found, Qt-DAB will report that and wait for another attempt to select a device. These precompiled versions (i.e. Windows installer(s) and AppImage) can be found in the Releases section of the github repository ("https://github.com/JvanKatwijk/qt-dab").

Of course, since the sources for Qt-DAB are available, one might compile an executable. Since building an executable from the sources is far from trivial, a separate document exists with a detailed description on what to do.

3 The GUI: the widgets and the control

3.1 Introduction

When starting the software for the first time, there is obviously no device selected yet. Therefore Qt-DAB shows two widgets, the *main* widget, and the so-called *configuration and control* widget. The latter shows at the right side of the bottom line a selector for one of the configured deviced. Once a device is selected the software starts decoding.

3.2 The widgets

The main widget shows the relevant data for the user's selection of a channel and a service. It displays slide(s) carried in the selected service, and the dynamic label text. Furthermore, it provides buttons for controlling the visibility of the *configuration and control* widget, the *spectrum widget* and the *technical details* widget, and for operations like scanning, scheduling etc.

Figure 1: main widget

The left side of the widget is dedicated to channel and service selection. It shows the *ensemble display*, in this view it shows the services of the current ensemble, changing the view with the button, here labeled *favorites* shows the list of favorites.

In the ensemble view the services also belonging to the favorites list are marked, in the favorites view the services have as indiction the channel where they were found.

Selecting a service in either view is by clicking with the mouse on the service name. The semantics of clicking with the mouse on the field right of the service name depend on the view mode: in ensemble view the service will be added to or removed from the favorites, in favorites view the service will be removed from the favorites.

With the channel selector, in the picture the small combobox labelled "1C", a channel may be selected, channels are labelled "5A" to "13F".

The *scanlist* button controls the visibility of the scan list, i.e. the list of services detected at the most recent *single scan*.

As mentioned, the button - here labeled *favorites* is used to switch views from *ensemble view* to *favorites view* and back.

Note that the *configuration and control widget* provides selectors for setting the *font*, the *font size* and the *font color* with which the service names are displayed.

Selectors If an ensemble is detected, the name of the ensemble appears above the servicelist ("NPO (8001)" in the picture above). *Clicking on that name* controls the visibility of the content widget, i.e. a widget showing the details of the services belonging to the current ensemble.

Audio services usually carry one or more slides that are displayed. In case a slide is

not (yet) found, a default slide is displayed.

When a service is selected, its name appears in bold on the right half of the widget. Left of that name ("NPO Radio 5" in the picture above) a small icon controls the visibility of the so-called *technical widget*, a widget showing details of the currently selected service.

The icon to the right of the name, the speaker, indicates whether or not the system produces sound. Furthermore, it acts as *mute* button.

Most services, when running, produce next to audio and one or more slides, text, the so-called DLS (Dynamic Label), "Rob de Nijs - Dag Zuster Ursula" in the picture above. Clicking with the right mouse button on this text shows a (very) small menu opening the possibility of saving the text to the OS' clipboard.

At the bottom of the widget one sees - if configuraed and available - the name of and dictance to the transmitter whose signal is being processed ("Alphen aan de Rijn/Celinex Toren Energieweg 19km 49" in the picture).

Finally, the 4 (four) buttons above this text control (from left to right)

- the visibility of the *configuration and control* widget, a widget with a huge amount of selectors for a large variety of setting of the system;
- the visibility of the *spectrum* widget, a widget dedicated to showing all kind of aspects of the incoming DAB signal;
- the *http handler*, a handler that controls showing a map with indications of the transmitters being received;
- the visibility of the *scan monitor* widget, a *separate* widget for controlling (single or multiple) scans over all or selected channels in the band.

3.3 Technical details

Most services are audio services, and it is always interesting to see what the parameters of the audio data are. The technical widget, see figure 2. shows these parameters.

On top of the widget, the name of the current service is repeated, together with - between brackets - the short name - if any. Below this line, there are two buttons, both for saving audio data of the current service. The button *frame dump* - when touched - instructs the software to dump the AAC frames of the audio service into a file. Such a file can be played by e.g. VLC. The button *dump audio* - when touched - instructs the software to dump the audio output into a ".wav" file, i.e. a PCM file, with a samplerate of 48000 samples/second.

The bottom of the widget shows the spectrum of the audio output, it shows that the audio frequency goes to app. 15 to 16 KHz.

The three progress indicators above the spectrum display (for MP2 output there will be only one) show the successrate of the different steps in the transformation from decoded DAB data to audio. The steps are:

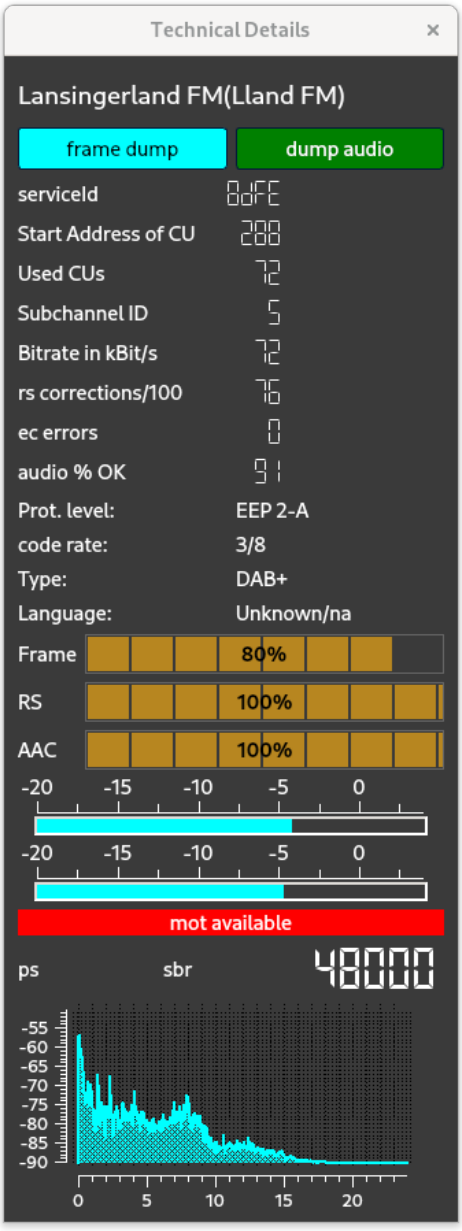


Figure 2: Technical details

- *Frame recognition and extraction.* DAB+ audio is organized in frames, recognizing frames in the input stream requires some testing and verification;
- *Reed Solomon* decoding. The indicator tells the successrate of the Reed Solomon decoder over the last 25 frames (Note however, that the R-S decoding can fix up to 5 errors per frame);
- *AAC decoding.* The indicator tells the successrate of interpreting the AAC frames and transforming them into audio (PVM) samples.

Most of the other data that is displayed speaks for itself, a few less obvious numbers:

- *rs corrections* displays the number of corrections, performed by the Reed-Solomon decoder in the last 100 frames. The Reed Solomon decoder operates with frames of 120 bytes, the last 10 of them being parity bytes, and is able to correct up to 5 byte errors in a frame;
- *ec errors* displays the number of errors detected *after* the Reed Solomon decoder has repaired the errors (of course, if the parity bytes in the data contain errors, it is hard to see how an error free audio frame can be constructed);
- *audio % OK* displays the percentage of audio frames that reaches the audio output. If the AAC decoding fails, no data is sent to the audio output (usually the soundcard).
- The *red label* with text "MOT" shows that the currently selected services does not carry MOT dat, i.e, the service described here does not contains slides and Qt-DAB will show some generic default slide.

3.4 The spectrum widget

As mentioned, in this version the different scopes (displays) are put into a single widget. The "tab" defines which scope is shown. The widget further contains an IQ scope, a waterfall display and a list of quality indicators. Depending on the configuration, there are 5 or 6 scopes, the fifth scope tells about the channel, the 6-th scope about the frequency offsets of the carriers in decoding.

The waterfall display is directly coupled to the selected scope, it will show the progress in time of the data of the selected scope. Figure 3 shows the spectrum of the signal. which has a width of just over 1.5 MHz. The number on the widget, 227.360, shows the frequency (in MHz) of the signal.

The "cross" in the box top right, is the so-called *constellation* of the *decoded signal*. The decoded signal elements are internally represented as complex numbers, and in the IQDisplay depicted in a two dimensional space. Ideally one sees 4 dots, one in the middle of each quadrant. In the examples here, the IQ display shows that while the phases differ app 90 degrees, the amplitudes of the different signal elements vary. (By clicking with

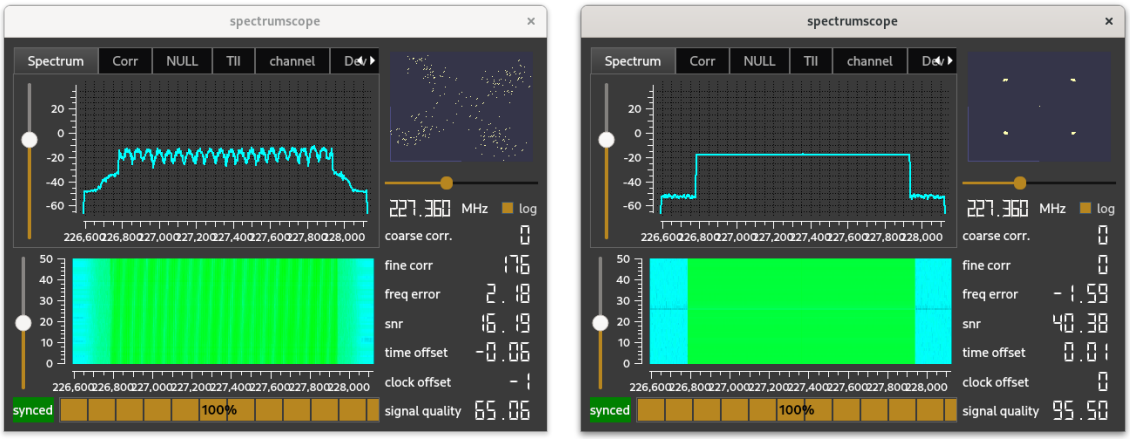


Figure 3: Spectrum scope and ideal signal

the right hand mouse button on the IQ dsplay, a switch is made to the constellation of the *input* rather than decoded, signal).

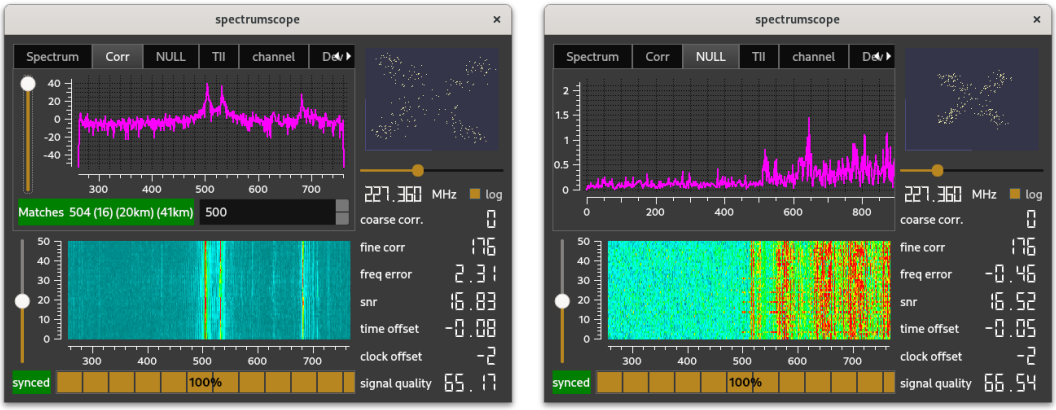


Figure 4: Correlation scope and NULL signal

The first step in decoding DAB signals is *synchronization*, i.e. finding where the actual data of a DAB frame starts in the incoming sample stream. The NULL scope (see figure 4, scope right) shows the samples in the transition phase from the NULL period to the first data block. To achieve such a synchronization a form of *correlation* is used (figure 4, scope left).

Ideally the maximum in the correlation is on (or about) sample 504 of the current segment of the incoming stream. The correlation scope here shows more than one peak, indicating that more than a single transmitter is received. The software detects here three "reasonable" peaks, it is up to the software to synchronize with the strongest signal.

If the current transmitter is "known", i.e. we know its name and distance, the

distance and the estimated distances of other peaks to the receiver site are shown. In the picture here, the transmitter transmitting the strongest signal has a distance of 19 km, the others, 16 km resp. 40 km to the receiving location

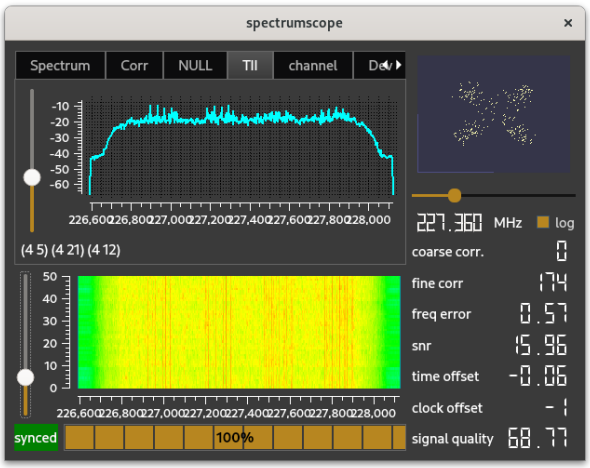


Figure 5: TII scope

As known, most transmitters send some identification data, encoded in the NULL period of the DAB frames. This data, Transmitter Identification Information (TII) consists of two numbers, a main Id and a subId, unique for each transmitter in a given country. The TII scope shows the spectrum of the NULL period, that is where the TII data is to be found. In this case, the software was able to detect that the TII data here can be decoded as (04 05). The so-called *mainId* and *subId* (04 05) belong to a transmitter in Rotterdam.

A 5-th scope, a so-called channel scope (see figure 7) shows the channel response.

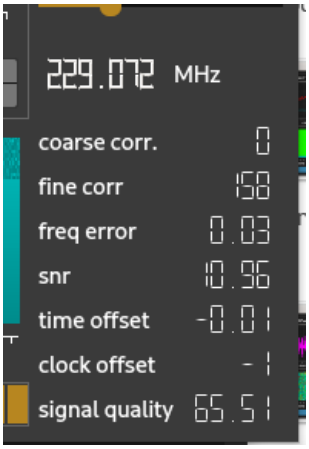


Figure 6: quality indicators

The first data block of a DAB frame has predefined data, so an estimate of the "channel" is made by looking at the transformation of a selected set of carriers between transmitter and receiver. The yellow coloured line in the picture shows the differences in amplitude, the red line the phase offset[1].

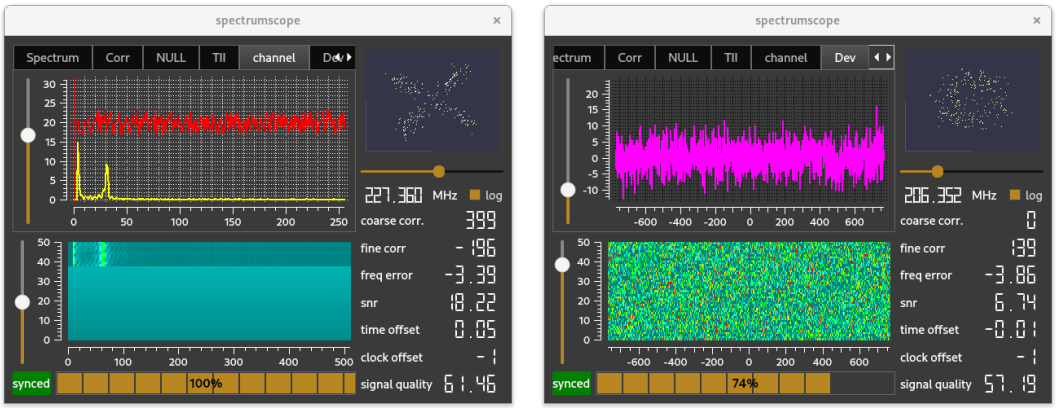


Figure 7: channel scope and deviations scope

The *deviation scope* shows the offsets, expressed in Hz, computed from the phase offsets in the decoded carriers. Ideally the offset in the decoded carrier is zero. Higher offsets means higher chances on errors in the decoding.

The widget shows a number of quality indicators, derived from the incoming signal. The numbers, an example shown in figure 6, read from top to bottom:

1. The *coarse frequency correction* and the *small frequency correction*, computed from a DAB frame and applied to the data for the next DAB frame. Qt-DAB is able to correct frequency errors to up to 25 Khz, here the error is pretty small. Using a "dabstick" or so as device will show a much larger correction.

2. the *remaining frequency offset*, which - obviously - should be small, but shows the effect of correcting incoming samples with an offset found earlier;

3. the measured *signal/noise ratio* (measured by looking at the signal strength in the NULL period and the period that data is transmitted);

4. the *time offset*, i.e. the error in the sample clock;

5. the *clock offset*, here we measure how many samples we are short (in this case) or we have too many per second.

6. the *quality* of the decoded signal, compared to the "ideal" signal that shows 4 dots, one in each quadrant. Higher values indicate a higher signal quality. The values

[1]A DAB block counts 1536 carriers, so the picture only shows a fraction

are scaled from 1 to 100, so the value above 60 shows a reasonable signal. There are many ways to compute a quality value, the approach in Qt-DAB is derived from ETSI RT 101 290, which is not completely fair since the decoding for DAB is primarily depending on the phase of the signal and less on the amplitude.

Finally, the bottom line contains a *label* and a *progressbar*. The label turns *green* if *time synchronization* can be achieved (i.e. the software "knows" where to find the DAB frames in the incoming sample stream.) The *progressbar* tells the success rate of decoding the FIC (Fast Information Channel) data. If the percentage is less than 100 percent, then apparently something is wrong with the signal and successfull decoding the payload is highly unlikely.

3.5 Configuration and control

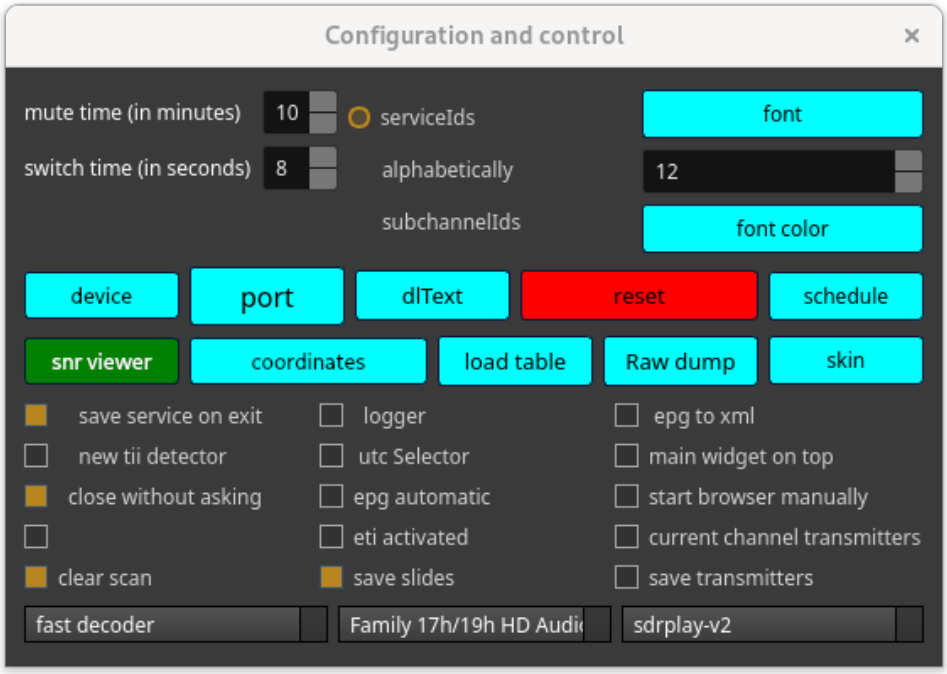


Figure 8: Configuration and control

The configuration and control widget is the widget with the buttons and checkboxes for influencing the configuration of the decoding process. The settings - together with other settings generated by the system - are stored in the ".ini" file.

- *mute time* tells the system what the duration should be in muting the audio;

- *switch time* tells the system what the waiting time should be after switching a channel before the software is convinced that the channel does *NOT* contains DAB data;

- *service order* tells the system in what order the services should be displayed on the main widget;

At the top right, there are two button and a spinbox. The *font*, the *font size* and the font color with which the service names are displayed on the main widget can be set.

Below this, there are two rows with push buttons

- *device* is a button controlling the visibility of the device widget; A device widget usually shows controls for setting e.g. gain, and once these are set, there is in general no need to have the widget visible;

- *port* allows setting *another port number* than the default one used otherwise for the http handler, i.e. for communicating with the webbrowser when displaying maps and data on maps;

- *dlText* controls an option to save the data from the *dynamic label* into a file. Touching it will show a file selection menu, touching it again will close the file. If a file is selected, the texts in the dynamic label are stored in the file. Note that in this version it is also possible to save the *current text* of the dynamic label. Select the section of the text with the mouse, click with the right hand side mouse button and a small menu appears with an option to save the selected text on the clipboard;

- *reset* will stop all activities and do a restart;

- *schedule button* - when touched - will make a schedule menu visible with which future actions (up to 7 days ahead(can be scheduled.

- *snr viewer* controls the visibility of a small display, showing graphically the progress in time of the SNR;

- *coordinates* allows specifying the local coordinates, used to compute distance and azimuth to transmitters.

- *load table*, if correctly configured, this allows (re)loading a new instance of the TII database (see section 4);

- *Raw dump* (Obsolete) allows selecting a file to which the input data will be saved in a PCM file. Use xml files instead.

- *skin* allows selection of a skin for the displayed widgets. *The selection will be effective the next program invocation*; The default skin chosen is *Darkeum*.

Below these buttons there is a list of 14 check boxe (in the column left one entry is unused)

- *save service on exit* tells the system to save the channel and service name on program termination, and to use these on program start up to initialize the system;
- *logger* obsolete;
- *epg to xml* tells the system to map EPG data using an imported library to xml. Note that it is known that the imported library has a bug that might lead to a crash of the program;
- *new tii detector*. TII detection may use a second algorithm that detects TII data sometimes earlier, but generates more erroneous TII values;
- *utc Selector*, when enabled shows time as UTC rather than local time;
- *main widget on top*, as it suggests, enabling this ensures that the main widget *always* will be on top. Note that it might cause problems on Windows, since newly generated widgets will be on top and therefore be covered by the main widget. Not ideal if the new, covered, widget asks for confirmation;
- *close without asking* does what it suggests;
- *epg automatic*. Some ensembles carry - next to regular services - an EPG (Electronic Program Guide) service. If enabled the system will execute that service in the background;
- *start browser manually*. By default, on enabling the http service, the local default browser will be started. If this option is enabled, the user has to start his/hers favorite browser;
- *eti activated*. Since Version 5 of the Qt-DAB software. there is an option of generating an *ETI*[2] *file* from the current channel input. If this option is emabled, the *scan* button on the main widget is changed in an "eti" button, with which the eti generator can be started and stopped;
- *current channel transmitters* if enabled, the transmitters shown on a map will only be the ones belonging to the current channel;
- *clear scan*. On scanning (single scan, see below) the service names envountered will be added to the scanlist that can be made visible on the main widget. If this option is activated, that list will be cleared on a new scan;
- *save slides* does what it suggests;
- *save transmitters*. If checked, transmitter names will be saved (see section 4).

[2]ETI stands for Ensemble Transport Interface, defined in ETS 300 799

bottom line The bottom line of the configuration and control widget contains three selectors, seldon the value of which is not modified often.

- a selector for the *decoder*, just to experiment a number of (more or less) different approaches are exercised to map the phase, resulting from the first decoder step, onto (soft) bits;
- the *audio out selector*, the selector is filled by the underlying sound system;
- the *device selector*, the configured devices are listed here. Note that Qt_DAB is unique in that during a session another device can be selected.

The settings of these selectors is always stored in the ".ini" file and used in initialization.

3.6 Coloring buttons and scopes

Colors for buttons and the scopes are personal. While not essential for functioning, the ability to choose one's on color scheme seems important to me. So, a color for the different buttons on the *main widget* and on the *configuration and control widget* can be set (and subsequently changed). Touch the button with the *right mouse button* and a small widget appears in which first the background color, and second the text color can be selected (see figure 9.

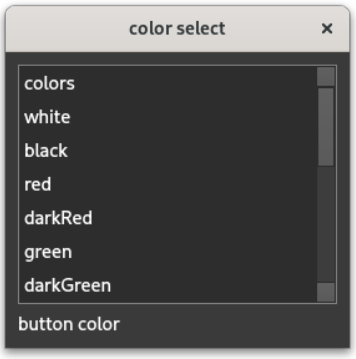


Figure 9: Button color selection

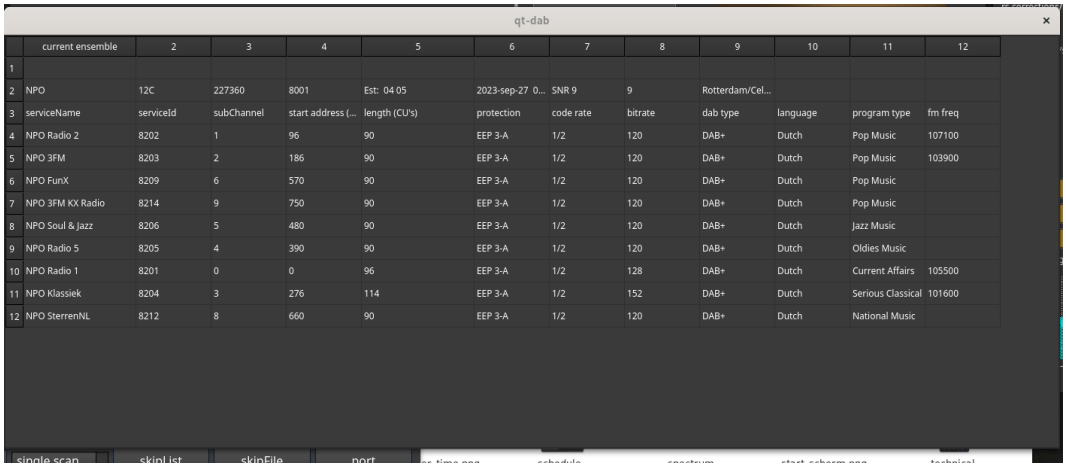
The same applies to the scopes in the spectrum widget, clicking with the *right mouse button* on the field will show a small selection widget with which a *background color*, a *grid color* and a *curve color* can be set. The color settings are immediately effective and maintained between successive program invocations.

4 Contents, scanning and maps

4.1 Introduction

Qt-DAB can display a description of the content of the currently selected channel and offers extensive options for scanning.

4.2 Looking at the content of the current channel



	current ensemble	2	3	4	5	6	7	8	9	10	11	12
1												
2	NPO	12C	227360	8001	Est: 04 05	2023-sep-27 0...	SNR 9	9	Rotterdam/Cel...			
3	serviceName	serviceId	subChannel	start address (...	length (CU's)	protection	code rate	bitrate	dab type	language	program type	fm freq
4	NPO Radio 2	8202	1	96	90	EEP 3-A	1/2	120	DAB+	Dutch	Pop Music	107100
5	NPO 3FM	8203	2	186	90	EEP 3-A	1/2	120	DAB+	Dutch	Pop Music	103900
6	NPO FunX	8209	6	570	90	EEP 3-A	1/2	120	DAB+	Dutch	Pop Music	
7	NPO 3FM KX Radio	8214	9	750	90	EEP 3-A	1/2	120	DAB+	Dutch	Pop Music	
8	NPO Soul & Jazz	8206	5	480	90	EEP 3-A	1/2	120	DAB+	Dutch	Jazz Music	
9	NPO Radio 5	8205	4	390	90	EEP 3-A	1/2	120	DAB+	Dutch	Oldies Music	
10	NPO Radio 1	8201	0	0	96	EEP 3-A	1/2	128	DAB+	Dutch	Current Affairs	105500
11	NPO Klassiek	8204	3	276	114	EEP 3-A	1/2	152	DAB+	Dutch	Serious Classical	101600
12	NPO SterrenNL	8212	8	660	90	EEP 3-A	1/2	120	DAB+	Dutch	National Music	

Figure 10: Content description

Of course, even without scanning, it is possible to get a view on what services are available in the currently selected channel. Next to the obvious list of service names as given on the ensemble display on the main widget, one might want a look at the "content", i.e. details of the ensemble and its constituents.

Picture 10 shows the content of the NPO ensemble, it shows that the TII identifiers at the time of reception were 04 05, the transmitter is located in Rotterdam, that 9 services were detected and the SNR was 9.

For each of the services (NPO does not transmit a data service anymore) the known data is printed. Note that it might take some time before all data items are known. Whether or not the service description shows an FM alternative FM frequency might not be known before a few seconds have passed.

4.3 Scanning

Touching the *scan* button controls the visibility of the scan monitor (figure 11. The *start* and *stop* buttons on the monitor widget have their obviosu functions, the button labeled *show* button controls the visibility of the *skiplist*.

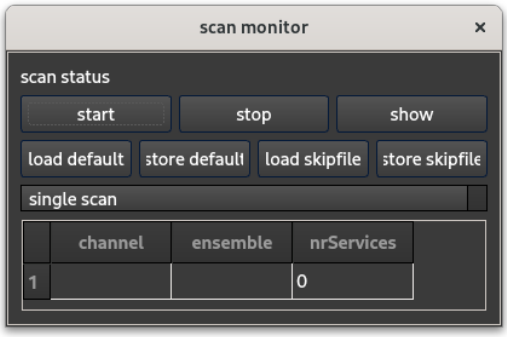


Figure 11: Scan monitor

In most cases it is known that on some specific channels no DAB signal will be found. Of course, it is a waste of time to include such a channel in the scan, therefore Qt_DAB has the possibility of indicating which channels can be skipped, hence the name *skipList*.

A default skiplist may be used, this list is stored somewhere with other DAB data. If, however, one wants different skiplists, e.g. for scanning in different directions, one may create separate - named - skiplists. Such a skiplist is stored in xml format in a file.

Scanning itself is in one of three modes:

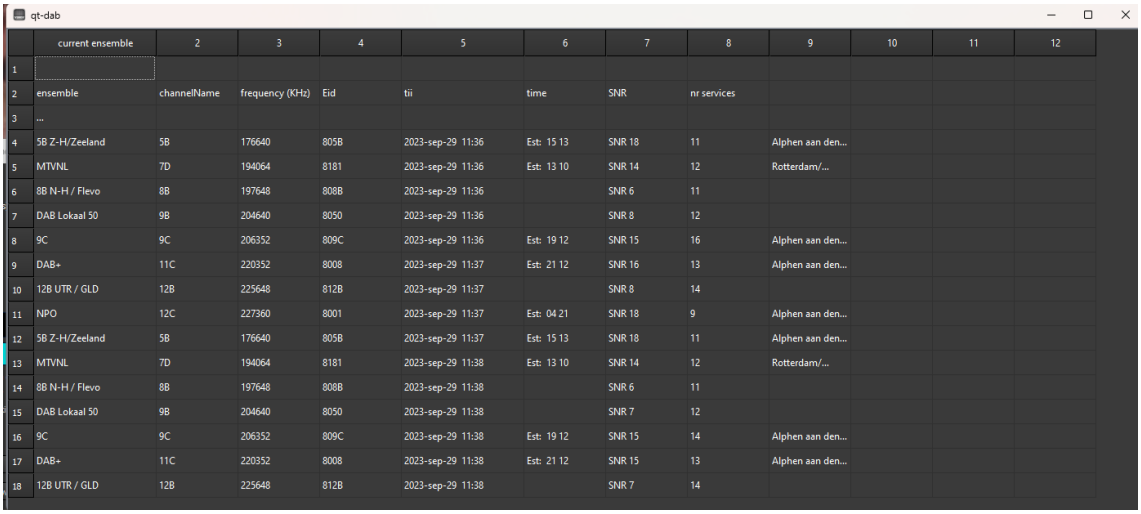
- in *until data* mode, scanning continuous until a channel is detected with DAB signals;
- in single scan mode - the default mode - a single scan is made over all channels (except of course the ones mentioned in the active skipList) of the band, and will stop afterwards;
- in *continuous* mode, scanning will continue until stopped by the user.

Of course, the output of the different modes will be different, while in the second mode a list is generated with for each channel that carries DAB data a full description of that data (similar to the content description), when running in continuous mode, only a single line will be generated for each ensemble encountered (see figure 12).

4.4 The TII database, transmitter names and distances

As mentioned earlier, the NULL period preceding the data in a DAB frame may contain an encoding of a number identifying the transmitter. DAB is transmitted with a large number of lower power transmissions, all using the same frequency, the encoding of the TII (Transmitter Identification Information) is added to the NULL period.

A large database exists (and is continually updated) with program and location information of DAB transmitters (see www.FMList.org) worldwide. The owner of the database kindly provided an URL to extract the relevant data for DAB transmitters for



qt-dab

	current ensemble	2	3	4	5	6	7	8	9	10	11	12
1												
2	ensemble	channelName	frequency (KHz)	Eid	tii	time	SNR	nr services				
3	...											
4	5B Z-H/Zeeland	5B	176640	805B	2023-sep-29 11:36	Est: 15 13	SNR 18	11	Alphen aan den...			
5	MTVNL	7D	194064	8181	2023-sep-29 11:36	Est: 13 10	SNR 14	12	Rotterdam/...			
6	8B N-H / Flevo	8B	197648	808B	2023-sep-29 11:36		SNR 6	11				
7	DAB Lokaal 50	9B	204640	8050	2023-sep-29 11:36		SNR 8	12				
8	9C	9C	206352	809C	2023-sep-29 11:36	Est: 19 12	SNR 15	16	Alphen aan den...			
9	DAB+	11C	220352	8008	2023-sep-29 11:37	Est: 21 12	SNR 16	13	Alphen aan den...			
10	12B UTR / GLD	12B	225648	812B	2023-sep-29 11:37		SNR 8	14				
11	NPO	12C	227360	8001	2023-sep-29 11:37	Est: 04 21	SNR 18	9	Alphen aan den...			
12	5B Z-H/Zeeland	5B	176640	805B	2023-sep-29 11:37	Est: 15 13	SNR 18	11	Alphen aan den...			
13	MTVNL	7D	194064	8181	2023-sep-29 11:38	Est: 13 10	SNR 14	12	Rotterdam/...			
14	8B N-H / Flevo	8B	197648	808B	2023-sep-29 11:38		SNR 6	11				
15	DAB Lokaal 50	9B	204640	8050	2023-sep-29 11:38		SNR 7	12				
16	9C	9C	206352	809C	2023-sep-29 11:38	Est: 19 12	SNR 15	14	Alphen aan den...			
17	DAB+	11C	220352	8008	2023-sep-29 11:38	Est: 21 12	SNR 15	13	Alphen aan den...			
18	12B UTR / GLD	12B	225648	812B	2023-sep-29 11:38		SNR 7	14				

Figure 12: Result from continuous mode

use with Qt-DAB only. Due to the latter requirement, the code to access the database can not be open source, and is therefore not part of the source tree.

However, code is included in the precompiled version with which the database can be loaded. The "load table" button on the configuration and control widget will ensure that a copy of the freshly acquired database is installed in the user's home directory.

Qt-DAB can also be configured to use a library with which accessing a local copy of the database is possible, that is why a local copy of the database is provided for in the source tree.

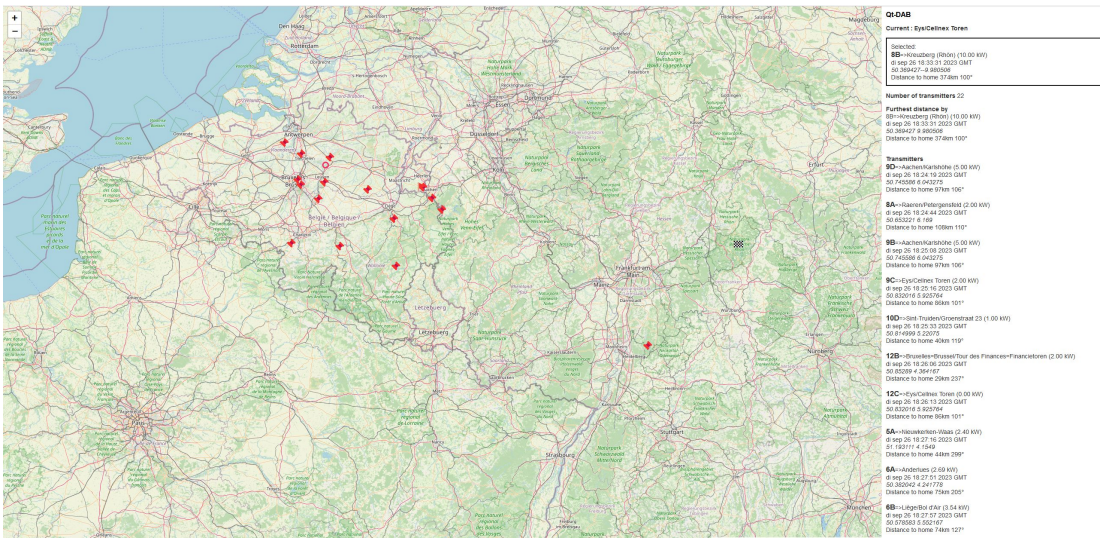


Figure 13: Transmitters on the map

Given that access is obtained to a local copy of the database, and given that the

home position of the system where Qt-DAB is running is known, Qt-DAB displays the name of the transmitter received (by looking up the TII code in the database), and will compute an estimate of the distance and the azimuth.

4.5 Transmitternames and a map

Qt-DAB offers the possibility of displaying a map on a webbrowser with the home location and the names and locations of the transmitters seen

The button *http* on the main widget will - whenever the database is accessible and the home location is known - issue a command to start a web browser, and will send data about the home location and the transmitters received to the browser program as shown in figure 13[3]. Note that while by default the *default* browser on the syste is selected, the *configuration and control* widget contains a checkbox that. when set, tells the software that the user wants to be able to select the webbrowser him/herself.

The right hand side of the map contains a list of transmitters, with for each transmitter the name, the precise location and if known the transmission power. The description is augmented with the distance and the azimuth from the specified home to the transmitter. In figure 14 part of the list is shown in more detail.

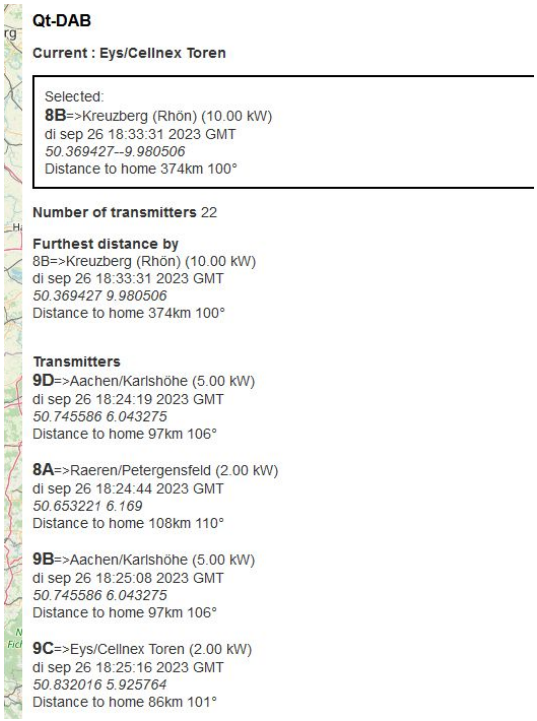


Figure 14: list of transmitters

[3]The picture of the map is made available by Herman Wijnants

The picture shows that the transmitter with the furthest distance was found on 374 Km, that the number of transmitters - collected for different channels - is 22.

The transmitter descriptions can be saved in a file. If the checkbox *save transmitters* is checked, the http handler will - on start up - show a file selection menu, for selecting a file in which the descriptions are stored.

5 Scheduling in Qt-DAB-6

When working, I am usually listening to a service that transmits music without too much talking. But then I want to hear the news bulletins on the hour on another service and of course in 9 out of 10 cases I was late. That is why Qt-DAB has a mechanism to switch over to a specified activity at a soecified times.

The *scheduling* option allows scheduling an operation to be performed within the next 7 days on a specified time. *Since the scheduling mechanism is part of the program, Qt-DAB should run for the scheduling to be effective.* Scheduling settings are kept between program invocations, on program startup, the software will remove schedule instructions that refer to the past.

Touching the schedule button shows a selection widget, as shown in figure 15

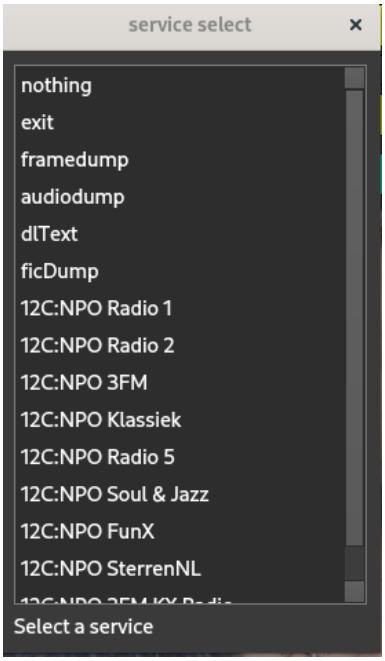


Figure 15: Schedule selection

The first step is selecting a service name or operation. The service names are those in the currently selected channel, and those in the preset list. Furthermore, some operations can be selected

- *nothing*, with the obvious semantics;
- *exit*, which, when executed will terminate the execution of Qt-DAB;
- *framedump* for scheduling the recording of the AAC frames for the service *that is active at the moment the scheduling time is reached*;
- *audiodump*, the same for the PCM output;
- *dltext*, for scheduling the start of saving the dynamic label text in a file;
- *ficDump*, for scheduling the start of the dump of the FIC data into a file.
- any of the services known to the software.

Note that executing the command when a previous invocation of that command is still active, will terminate the execution of the operation.

Having selected an operation of service, the next step is specifying the time and day of the operation to be executed. A small widget shows, that allows setting day and time. see figure 16

If the list of scheduled events is not empty, it will be shown in a separate (small) widget, see figure 16.

6 Supported input devices

Qt-DAB supports a variety of input devices, the Adalm Pluto, the AIRspy, the hackrf,

Figure 16: Schedule time selection and the schedule list

the limeSDR, RT2832 based sticks and SDRplay RSP devices. Furthermore, there is support for the rtl_tcp server, support for the spyserver, for file input (raw, wav and xml), and for devices for which a *Soapy* interface library exists,

Both the *appImage* and the *Windows installer* are configured with (almost) the whole range of devices: SDRplay RSP (different versions for the 2.13 and 3.XX library versions), the Adalm Pluto, the AIRspy, the hackrf, the LimeSDR, and - of course - the RT2832 based dabsticks.

6.1 The SDRplay RSP

The Qt-DAB software supports all RSP's from SDRplay. Qt-DAB provides two different device handlers for the RSP's, one for devices using the 2.13 SDRplay interface library, the other one supports devices using the 3.0X SDRplay interface library. Note that if API 3.10 or up is installed on Windows, the 2.13 library is not accessible. Note furthermore that the recently announced RSP 1B is supported.

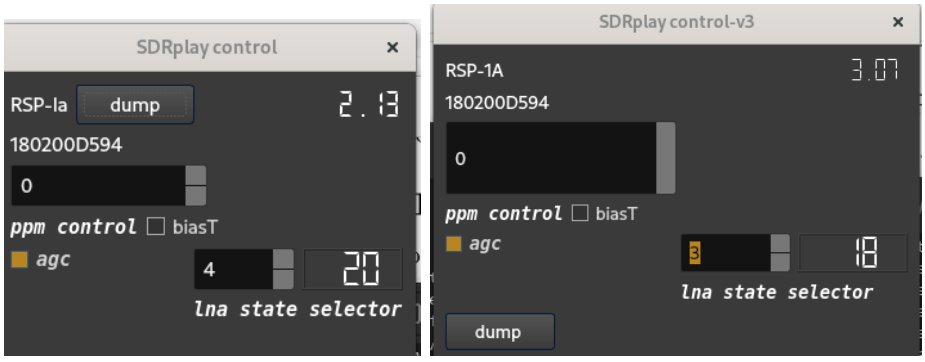


Figure 17: Qt-DAB: The two control widgets for the SDRplay

As figure 17 shows, the control widgets for the two different versions resemble each other, their implementation differs considerably though. Both have spinboxes for setting the *if gain reduction*, the *lna state* and a *ppm offset*.

Since Qt-DAB is capable of correcting th frequency, there is no actual need for setting a value here.

The spinbox for the *if gain reduction* is programmed to support the range of values between 20 and 59. The range of values for the *lna state* depends on the model of the RSP. The software will detect the model and fill in the range accordingly.

If the *agc* is selected, the *if gain reduction* spinbox will be hidden, its value is then irrelevant.

The RSPdx (and its predecessor, the RSP II) has two (actually 3) slots for connecting an antenna. If an RSPdx or RSP II is detected, a combobox will be made visible for *antenna selection*.

Finally, both versions of the control widget contain a *dump* button. If touched, the raw input from the connected device will be stored in a so-called *xml formatted* file. First a menu is shown for selecting a filename, a suggestion for the name of the file *device name - date* is given. Touching the button again will stop dumping and the file will be closed.

If more than one connected device is detected, a widget appears on which a selection can be made which device to use.

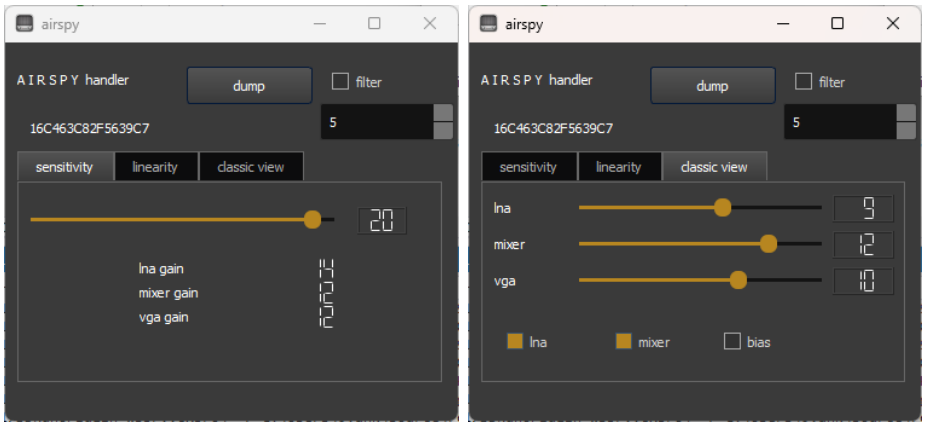


Figure 18: Qt-DAB: Widgets for AIRspy control

6.2 The AIRSpy

The control widget for the AIRspy (figure 18, right) contains three sliders and a push button. The sliders are to control the *lna gain*, the *mixer gain* and the *vga gain*.

To ease balancing the setting of the sliders, two combined settings are included in the widget, selectable by the tab *sensitivity* and *linearity*. Figure 18 left side, shows the setting at selecting the tab *sensitivity*.

Touching the button labeled *dump* instructs the software to dump the raw stream of samples into a file in the *xml format* (Note that while processing DAB requires the samplerate to be 2048000, that rate is *not* supported by the AIRspy, implying that the driver software has to do some rate conversion. The xml file though will just contain the samples on the rate before conversion).

If more than one airspy is detected, a widget will appear with which the device to use can be selected.

6.3 The Hackrf

The control widget for hackrf (figure 19) shows, next to the Serial Number of the device, a few sliders, a few checkboxes, a spinbox and a push button.

- the *sliders* are there for controlling the lna and vga gain, the slider values are limited to the range of possible values;
- The *Ant Enable* checkbox is for Antenna port Power control (not used in this controller);
- The *Amp Enable* checkbox is - if enabled - for additional gain on the antenna input;
- the *ppm correction* spinbox can be set to correct the oscillator (on 227 MHz, the Qt-DAB software reports an offset of somewhat over 3 KHz);

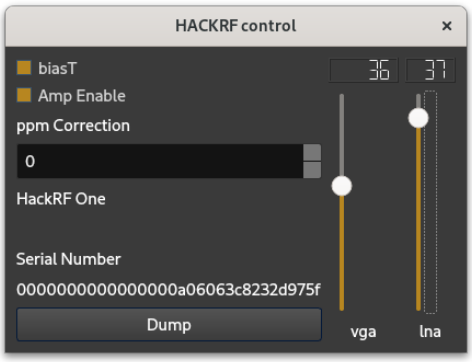


Figure 19: Qt-DAB: Widget for hackrf control

- the *Dump* push button when pushed, starts dumping the raw input in xml file format. Touching the button again will halt the dumping and close the file.

6.4 The LimeSDR

Figure 20: Qt-DAB: Widget for Lime control

On selecting the LimeSDR (if configured), a control widget for the LimeSDR is shown (figure 20). The widget contains five controls:

- *gain* control, with predefined values;
- *antennas*, where *Auto* is usually the best choice;
- *Dump*, if touched, the raw input from the connected device will be written to a file in the so-called xml format.

New is the inclusion of a filter. Note that the limeSDR reads samples with a samplerate of 204KHz with no filtering, while the frequency distance between successive channels is 1712KHz. So, DX-ing in adjacent channels, where e.g. the first channel contains a strong DAB signal and the next one a weak, is difficult.

Therefore the control widget for the limeSDR has two additional controls,

- switching a software FIR filter on-off (the checkbox labeled *filter*),
- setting the filterdepth of the FIR filter (the spinbox below the checkbox).

Note that using the filter is not free, for a filter with a size of N, N * 2048000 complex additions and multiplications are performed. While on a modern PC that is not an issue, it certainly is on ARM based micros like the RPI 3.

6.5 The RTLSDR stick

On selecting the dabstick (i.e. RT2832 based devices) (if configured), a control widget for the device appears (figure 21).

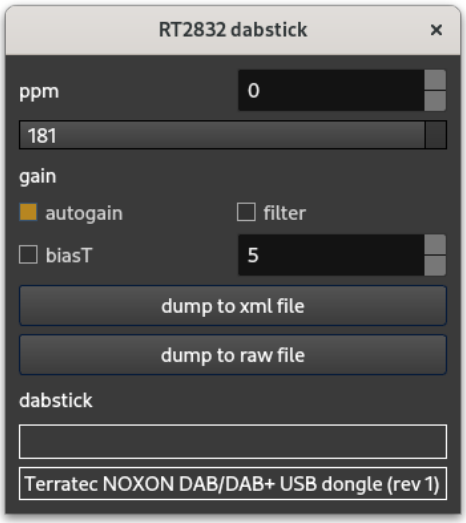


Figure 21: Qt-DAB: Widget for rtlsdr device

The widget contains just a few controls:

- a *spinbox* for setting the ppm. Note that on average the offset of the oscillator with DABsticks is (much) larger than that with devices like the SDRplay. The DAB software is able to correct frequencies to up to app 35 KHz, for some sticks the frequency error was large and correction using the ppm setting was required.
- a *combobox* for setting the gain. The support software for RT2832 based devices generates a list of allowable gain settings, these settings are stored in the combobox;

- a *combobox* for setting the autogain on or off;
- a *push button* that, when touched, will instruct the software to dump the raw input in the aforementioned xml format. At first a menu appears for selecting a file. Touching the button again will stop dumping and close the file.

New is the inclusion of a *filter*. Note that the DABstick reads samples with a speed of 2048S/s for a signal with a bandwidth of app 1.536 MHz, while the frequency distance between successive channels is 1712KHz. So, DX-ing in adjacent channels, where e.g. the first channel contains a strong DAB signal and the next one a weak, is difficult.

The controller therefore contains an optional FIR filter, for which the rtlsdr control widget has two additional controls:

- switching a software filter on-off (the checkbox labeled *filter*),
- setting the size of the FIR filter (the spinbox below the checkbox).

Note that switching the filter on is not for free, for a filter with a size of N, N * 2048000 complex additions and multiplications per second are performed. While on a modern PC that is not an issue, it certainly is on ARM based micros like the RPI 3.

If more than one connected RTLSDR based device is detected, a widget appears on which the device of choice can be selected.

6.6 The Pluto device

When selecting *pluto*, a widget (figure 22) appears with a spinbox for selecting the gain, and a checkbox for selecting the agc. If *agc* is enabled, the spinbox for the gain setting is invisible. The widget contains furthermore three buttons:

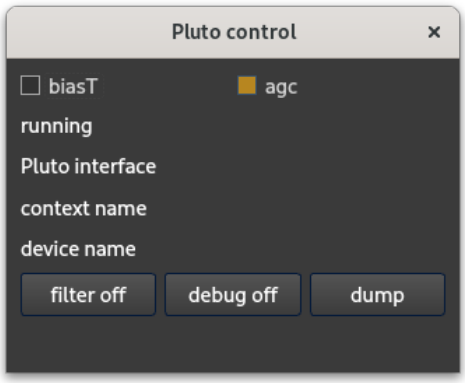


Figure 22: Qt-DAB: Widget for Adalm Pluto device

- the *debug control* button, when activated, instructs the software to show output on each step in the initialization process (note that the setting of the debug button will be maintained between invocations);
- the *dump* button will cause the original input - with a samplerate of 2100000 samples per second[4] - to be stored in an xml file.
- the *filter* button. The adalm pluto has as option specifying a fir-filter, to be executed within the Pluto device. This implementation of the controller for pluto will load a predefined filter onto the Pluto device which is enabled by default. With the filter button the filter can be disabled or enabled. Note that the button text indicates the action when touching, not the current state.

6.7 Support for Soapy (Linux only)

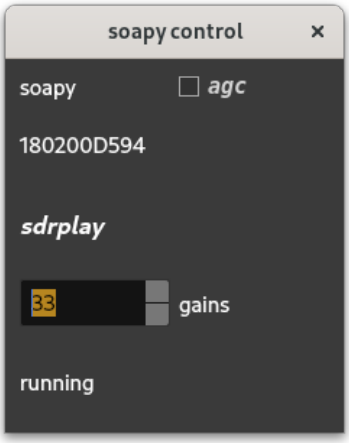


Figure 23: Qt-DAB: Widget for soapy

Soapy is a generic device interface, a kind of wrapper to provide a common interface to a whole class of devices. Qt-DAB supports Soapy, and its use is tested with the Soapy interface for SDRplay, hackrf and rtlsdr device.

The widget for soapy control (see figure 23) when applied to the soapy interface for the SDRplay contains simplified controls, compared to the regular controls for the SDRplay.

Note however, that to use Soapy for interfacing a specific device, a soapy device interface should have been installed for that device.

[4]The smallest samplerate that pluto gives is slightly larger than the required 2048000, 2100000 is chosen since it is easy to handle

6.8 rtl_tcp

rtl_tcp is a server for rtlsdr devices, delivering 8 bit IQ samples (i.e. 2 bytes per sample).

In the small widget (see figure 24) the ip address of the server can be given. Since the default port for the server is 1234, that port number is the one used by the client. After clicking the *connect* button, the client will look for a server and pass some parameters.

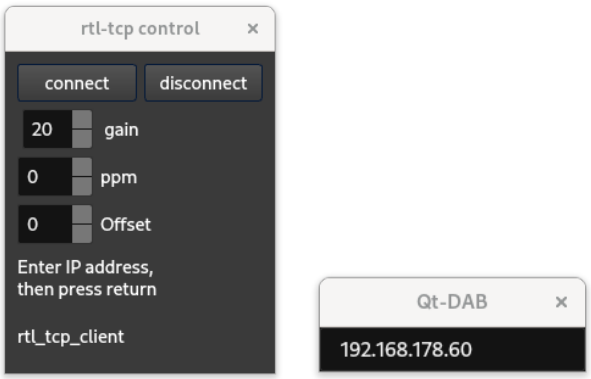


Figure 24: Qt-DAB: Widget for rtl_tcp

However, the port number can be set in the ".ini" file, by setting

```
rtl_tcp_port=XXX
```

where XXX is to be replaced by the portnumber of choice.

6.9 Input from a spyServer

Spyservers are well known, they provide a common interface to the remote use of either an AIRspy device or an RTL2832 based dabstick (see figure 25).

On starting the spy server interface asks for an IP address to be filled in, the port used is 5555.

Qt_DAB offers two versions of the spyserver, one for 8 bit output and one for 16 bit output. Of course, transmission of DAB input requires quite some bandwidth, for 16 bit data, and an input rate of 2500000 (as for the AIRspy), with 4 byte samples (2 x 2), the transmission rate is over 10 MByte/second. Since data is sent as packages, with a (small) header, the actual rate is slightly over 10 Mbyte. Of course, the using the 8 bit variant reduces it to app half.

6.10 File input

Qt-DAB supports both *writing* raw input files and *reading* them back. Writing a file as PCM file is initiated by the *Raw dump* button on the main GUI, writing a file as xml file by the *dump* button on the various device widgets. Qt-DAB differentiates between reading

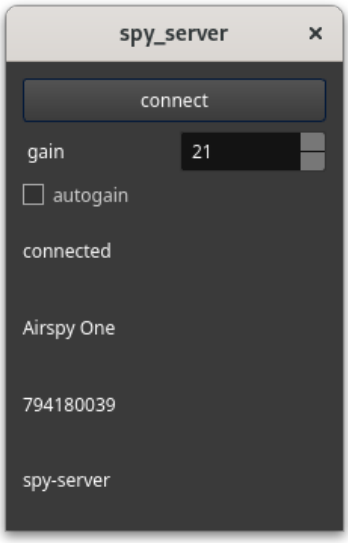


Figure 25: Widget for spyserver

- raw 8 bit files as generated by e.g. Osmocom software (usually files with an extension ".raw" or ".iq");
- PCM (i.e. ".wav") files, provided the data is 2 channels and with a samplerate of 2048000, generated by Qt-DAB and with an extension ".sdr";
- xml files. The xml file format was defined by Clemens Schmidt (author of QIRX) and me and aims at saving files in the original format, so to allow easy exchange between different DAB decoder implementations. In order to support proper decoding of the contents, the data in the file is preceded by a detailed description in xml, hence the name xml file format.

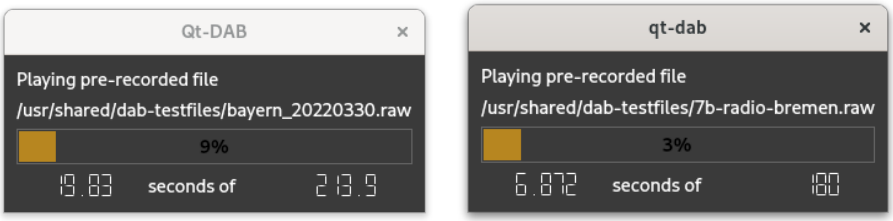


Figure 26: Qt-DAB: Widgets for file input

When selecting file input ".raw" or ".wav", a simple widget is shown (figure 26), with as indication the number of seconds the file is being played.

Since processing an xml file implies some interpretation, the widget (figure 27) for control when reading an xml file is slightly more complex. It contains - next to the

progress in reading the data - a description of the contents of the file. So, the program that generated the file as well as the device used in that program are displayed, the number of bits of the samples, as well as the number of elements is displayed as is the samplerate of recording and the frequency of the recording.

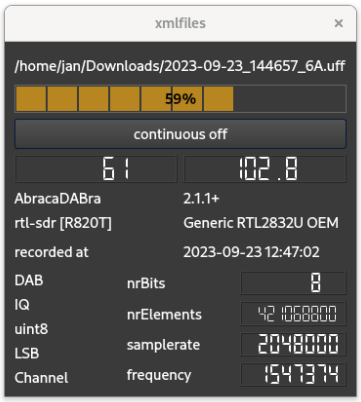


Figure 27: Qt-DAB: Widget for xml file input

Touching the *cont* button will instruct the software to restart reading at the beginning of the segment in the file after reaching the end.

6.11 Other devices

Interfacing another device to Qt-DAB is not that complex, the interface contains a handful of functions that are to be implemented. Interfacing is described elsewhere. The sourcetree of Qt-DAB contains code for a few device handlers, these handlers are - since I do not have access to the devices for which they are written - untested. While the uhd device seemed to have been working with the included device handler, the device handlers for both the elad-s1 and the colibri are very experimental and not tested. Anyway, I am always interested to experiment further with these devices.

7 Acknowledgements

Qt-DAB and derived programs are written and maintained by me. The software is provided *as is*, and made available under the Gnu GPL V2 license.

Many people contributed (and contribute) by providing feedback, suggestions and code fragments, in particular:

- Andreas Mikula, for continuous feedback, testing and suggestions;
- Herman Wijnants, for his continuous enthousiastic feedback on and suggestions for the Windows version of Qt-DAB;

- Stefan Pöschel, for providing code for and giving suggestions to handling the AAC code;
- Stuart Langland, for its comments, suggestions and code contributions;
- probonopd, for its contribution with creating appImages;
- Przemysław Wegrzyn, for contributing code for handling charsets;
- Paul Howard-Beard, for his enthousiastic experiments with new features, comments and his suggestion to add features like a mute button, the per channel gain settings, and alarm; and
- Michael Lass, for showing me the use of the Gcc address sanitizer, pointing out some (actually too many) address violations discovered by the sanitizer and giving suggestions and advice for the repair.

Furthermore I am grateful

- to SDRplay ltd (Andy Carpenter), for providing me the possibility to use the Ia and II versions of the SDRplay RSP devices, all wonderful devices;
- to Benjamin Vernoux, for making an AIRSPY device available;
- to Great Scott Gadgets, for making an HACKRF device available;
- to Jan Willem Michels, for making a LimeSDR device available;
- to Olaf Czogalla, for donating an RT2832 based stick after having lively discussions on TPEG; and
- to Robin Getz (Analog Devices), for making an Adalm Pluto available, a device with lots of possibilities, still to discover.

8 Disclaimer

Qt-DAB is developed as hobby program in spare time. Being retired I do have (some) spare time and programming Qt-DAB (and my other programs) is just hobby.

It is important to notice that Qt-DAB is - both the sources as the precompiled versions - available *as is*. While it is - obviously - nice if the Qt-DAB software is useful, *there are no guarantees, however*. If the software does not fit your purpose, it might just not be the software you are looking for, read the license, that states (a.o):

the Qt-DAB software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or

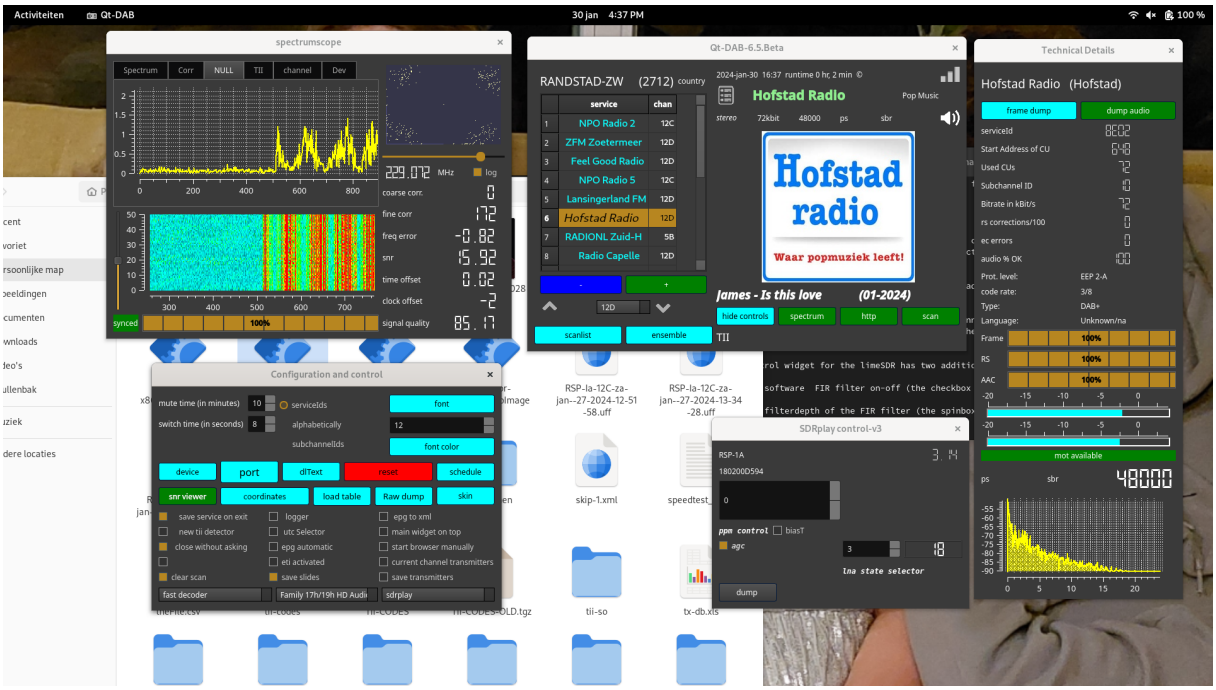


Figure 28: Qt-DAB and the 4 widgets

FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

If, however, the software seems useful and you have suggestions for improving and/or extending it, feel free to contact me, suggestions are always welcome (although no garantees are given that they will be applied.)